2부 기계학습을 이용한 텍스트 분석

지금까지 두 가지의 방법을 이용해서 텍스트 데이터를 분석해 보았습니다. 바로 빈도 분석과 텍스트 네트워크 분석이었습니다. 두 분석 방법 모두 전처리 과정을 거친 불용어가 제거된 특정한 품사 (주로 명사)의 단어들을 추출해서 분석에 사용했습니다. 지금부터는 기계학습 알고리즘을 이용한 텍스트 분석 방법에 대해 살펴보도록 하겠습니다. 기계학습 알고리즘을 이용해서 텍스트 분석을 제대로 수행하기 위해서는 기계학습 알고리즘의 작동 원리를 이해하는 것이 필요합니다. 따라서 본 책에서는 기계학습에 대해서 먼저 설명하도록 하겠습니다.

# 기계학습(Machine learning)에 대한 이해

## 기계학습이란?

기계학습(Machine learning)은 간단하게 말하면 기계, 즉, 컴퓨터가 데이터를 학습한다(A machine learns)는 것을 의미합니다. 좀 더 구체적으로 말하면 데이터가 가지고 있는 여러가지 정보들 중에서 주어진 문제를 푸는데 있어 중요한 역할을 하는 정보를 추출하는 것을 의미합니다. 그리고 이러한 학습에 사용되는 도구를 기계학습 알고리즘 (algorithm)이라고 합니다.

### 기계학습 알고리즘의 유형

기계학습 알고리즘은 일반적으로 지도학습, 비지도학습, 강화학습 세 가지로 구분됩니다. 그 중 강화학습 알고리즘은 사용되는 정도가 아직까지는 제한적으로 컴퓨터 게임 등 일부 분야에서 사용됩니다. 텍스트 분석에서는 사용되는 정도가 크지 않아 본 책에서는 지도학습과 비지도학습 알고리즘 중심으로 설명합니다.

**지도학습과 비지도학습 알고리즘**

① 지도학습 (supervised learning) 알고리즘

지도학습 알고리즘은 정답과 정답을 맞히는데 필요한 힌트 정보가 모두 있는 데이터를 학습하여 정답과 힌트 간의 관계를 파악하고, 그 결과를 풀고자하는 문제에 대한 데이터에 적용하여 주어진 문제를 해결하는 식으로 작동하는 알고리즘을 의미합니다.

② 비지도학습 (unsupervised learning) 알고리즘

비지도학습 알고리즘은 관측치들의 특성 정보를 담고 있는 데이터에 적용되어 데이터에 존재하는 패턴 혹은 인사이트를 찾는 목적으로 사용됩니다. 지도학습 알고리즘이 적용되는 데이터와 달리 데이터에 저장된 정보를 정답과 힌트 정보로 구분하지 않습니다. 비지도학습 알고리즘을 사용한 분석의 대표적인 예로는 군집화와 차원축소 등이 있습니다.

두 가지 종류의 알고리즘들 중에서 지도학습 알고리즘의 사용 정도가 더 많기 때문에 본 섹션에서는 지도학습 알고리즘을 중심으로 기계학습 알고리즘이 작동하는 방식에 대해 설명하도록 하겠습니다.

### 지도학습 알고리즘의 작동 원리

컴퓨터가 학습을 하기 위해서는 학습에 사용되는 데이터가 있어야 합니다. 이러한 데이터를 학습 데이터 (training data)라고 합니다. 지도학습에서 사용되는 학습 데이터에는 반드시 두 가지 종류의 정보가 모두 포함되어 있어야 합니다. 하나는 풀고자하는 문제에 대한 정답 정보이고 다른 하나는 정답을 맞히는데 필요한 힌트 정보입니다. 컴퓨터는 학습을 통해 학습 데이터에 존재하는 힌트와 정답 간의 (최적의) 관계를 파악하게 됩니다. 그리고 이러한 관계를 파악하는데 사용되는 도구가 (지도학습) 알고리즘이 됩니다. 알고리즘은 간단하게 생각하면 수학적 모형 (mathematical model) 또는 수학적 함수 (function)이라고 생각할 수 있습니다. 즉, 지도학습의 경우, 수학적 모형을 이용해서 학습 데이터에 존재하는 정답과 힌트 간의 관계를 파악하고, 그 결과를 풀고자하는 문제에 대한 데이터에 적용해서 주어진 힌트 정보를 이용해 정답을 예측하게 됩니다.

구체적인 예를 들어보도록 하겠습니다. 여러분들이 풀고자 하는 문제가 아파트의 크기 (평수) 정보를 이용하여 아파트의 가격을 예측하는 것이라고 가정합니다. 이러한 경우, 아파트의 가격이 정답이 되는 것이고 정답을 맞히는데 사용되는 아파트의 크기가 힌트가 됩니다. 아파트의 크기 정보를 이용해서 가격을 정확하게 예측하기 위해서는 아파트의 크기와 아파트 가격 간의 관계를 아는 것이 필요합니다. 그렇다면 둘의 관계를 어떻게 알 수 있을까요? 이러한 관계를 파악하기 위해서는 아파트의 가격 정보와 크기 정보, 즉, 정답과 힌트 정보가 담긴 학습 데이터가 필요합니다. 그림 7.1은 이러한 학습 데이터의 예를 보여주고 있습니다. 해당 학습 데이터에는 네 채의 아파트에 대한 정답과 힌트 정보가 있습니다. 예를 들어, 첫 번째 아파트의 크기는 30평이고, 거래 가격은 3억원입니다. 아래의 예에서는 한 채의 아파트가 하나의 관측치가 됩니다.

|  |  |
| --- | --- |
| 크기 (평수) | 가격 (억원) |
| 30 | 3 |
| 34 | 4 |
| 34 | 3.8 |
| 48 | 6 |

그림 . 학습 데이터의 예

컴퓨터는 이러한 학습 데이터 존재하는 힌트와 정답 간의 관계를 수학적 모형 (즉, 기계학습 알고리즘)을 이용해서 파악하게 됩니다. 그리고 그러한 과정을 학습이라고 하는 것입니다. 힌트에 해당하는 아파트의 크기를 독립변수[[1]](#footnote-2) (영어로는 independent variable, explanatory variable, predictor, regressor 등으로 표현합니다), 정답에 해당하는 아파트의 가격을 종속변수 (dependent variable)라고 합니다. 즉, 컴퓨터는 수학적 모형을 이용해서 학습 데이터에 존재하는 독립변수(들)와 종속변수 관계를 파악하게 됩니다.

그렇다면 학습에 사용되는 수학적 모형은 어떻게 생겼을까요? 수학적 모형은 다르게 표현하면 수학적 함수라고 할 수 있습니다. 여러분들이 알고 있는 1차 함수, 2차 함수 등의 다항함수, 지수 함수, 로그 함수 등이 그러한 예입니다. 수학적 모형을 이용해서 학습 데이터에 존재하는 힌트와 정답 간의 관계를 어떻게 파악할 수 있는지를 설명하기 위해서 가장 간단한 형태의 수학적 함수인 1차 함수의 예를 들어 보겠습니다. 독립변수가 하나인 경우에는 아래와 같이 표현됩니다.

보통 y는 X의 함수라고 하며, 라고 표현합니다. 1차 함수의 경우는 가 되는 것입니다. 그럼 에서 은 무엇을 의미할까요? 은 X = 0 일 때의 y값, 즉 y 절편을 의미합니다. 은 기울기를 의미합니다. 즉, 이 됩니다. 이 의미하는 것은 X의 값이 1만큼 증가할 때 y의 값이 얼마만큼 증가 또는 감소하는지를 의미합니다. 경우는 X가 증가할 때 y도 증가한다는 것을, 경우는 X가 증가할 때 y는 감소한다는 것을 의미합니다. 이러한 일차 함수는 그림 7.2와 같이 직선으로 표현됩니다.

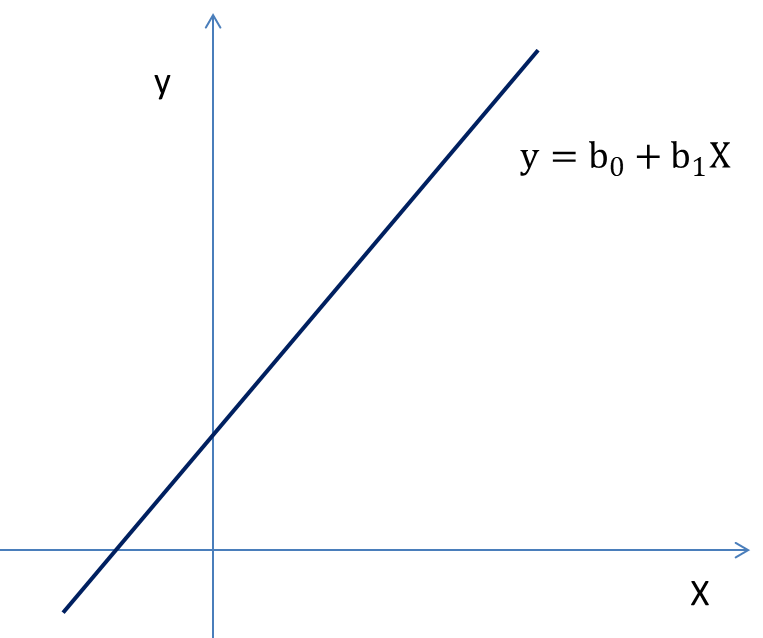


그림 . 1차 함수의 예

그렇다면, 은 어떠한 역할을 할까요? 그림으로 설명하자면, 가 구체적으로 어떠한 값을 갖느냐에 따라서 으로 표현되는 직선의 모양이 달라집니다. 예를 들어서 과 인 두 경우에 대해서 를 표현해 보면 그림 7.3과 같이 됩니다.

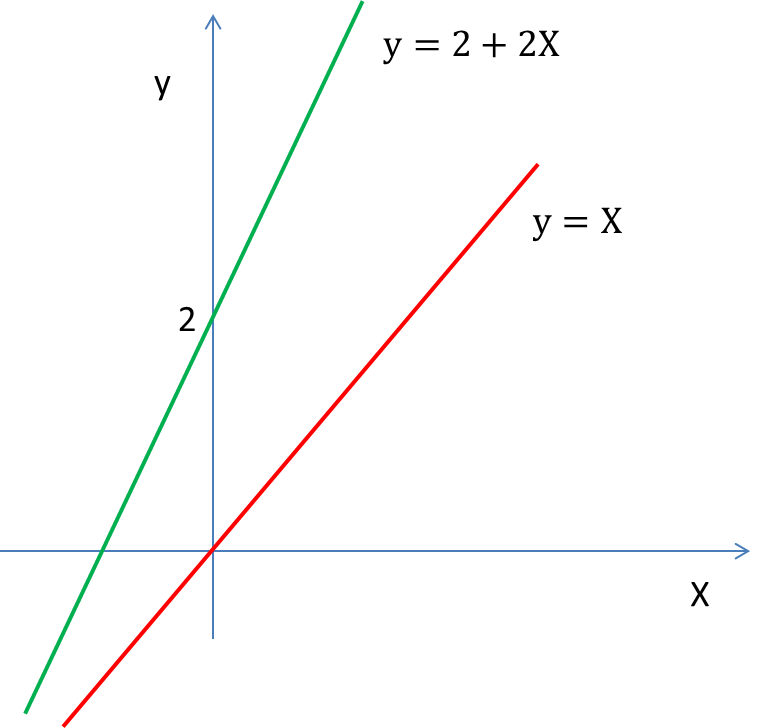


그림 . 와 의 값에 따른 직선의 형태

그림 7.3에서 보이는 것 처럼 의 값에 따라서 직선의 모양이 달라집니다. 그리고 구체적인 직선의 모양은 X와 y의 관계를 의미합니다. 즉, 1차 함수에 의해 설명되는 X와 y의 관계는 과 가 구체적으로 어떠한 값을 갖느냐에 따라서 달라지는 것입니다. 예를 들어, , 즉, 는 의 값이 0인 경우 의 값도 0이고, 가 1 만큼 증가할 때 의 값도 1 증가하는 관계를 나타냅니다. 반면에, , 즉, 는 의 값이 0일 때 의 값은 2가 되고, 의 값이 1 증가할 때 의 값은 2 증가하는 관계를 의미합니다. 이러한 의미에서 을 함수의 파라미터 (혹은 모형의 파라미터)라고 합니다. 파라미터는 수학적 모형에 의해 설명되는 독립변수와 종속변수의 관계를 정의하는 역할을 합니다. 즉, 파라미터가 구체적으로 어떠한 값을 갖느냐에 따라 독립변수와 종속변수의 관계가 달라지는 것입니다. 우리는 학습을 통해서 파라미터가 취할 수 있는 여러 값들 중에서 우리가 선택한 수학적 모형을 이용해서 독립변수와 종속변수의 관계를 가장 잘 설명하는 파라미터의 값을 찾게 됩니다. 이러한 파라미터의 값을 최적값 (optimal value)라고 합니다. 즉, **학습한다라는 것은 선택되어진 수학적 모형 (즉, 기계학습 알고리즘)이 갖는 파라미터의 최적값을 찾는다라는 것을 의미**합니다. 그리고 그렇게 찾아진 파라미터의 최적값을 사용해서, 독립변수에 대한 정보만을 가지고 있는 새로운 데이터에 존재하는 종속변수의 값을 예측하게 되는 것입니다.

그렇다면 파라미터의 최적값 (optimal values)는 어떻게 찾아낼 수 있을까요?

파라미터의 최적값은 우리가 선택한 수학적 모형 (즉, 기계학습 알고리즘)을 이용해서 학습 데이터에 존재하는 독립변수와 종속변수 간의 관계를 가장 잘 설명하는 값이라고 했습니다. 가장 잘 설명한다는 것은 다르게 표현하면 설명하지 못하는 정도가 최소가 된다는 것을 의미합니다. 기계학습에서는 파라미터의 최적값을 찾기 위해 모형을 이용해 학습 데이터에 존재하는 종속변수를 설명하지 못하는 정도를 최소화하는 파라미터의 값을 찾습니다. 그리고 모형이 종속변수를 설명하지 못하는 정도를 함수를 이용해서 나타내는데 이러한 함수를 비용함수 (cost function) 또는 손실함수 (loss function)이라고 합니다. 즉, 파라미터의 최적값은 비용함수의 값을 최소화하는 파라미터의 값이 됩니다. 일반적으로 비용함수의 종류는 풀고자하는 문제의 종류에 따라서 구분됩니다. 지도학습 알고리즘을 이용해서 풀 수 있는 문제의 종류는 크게 두 가지로 구분됩니다. 하나는 회귀 문제 (regression problem)이고 다른 하나는 분류 문제 (classification problem)입니다. 회귀 문제는 일반적으로 종속변수가 연속형 변수(예, 연봉, 매출 등)인 경우이고, 분류 문제는 종속변수가 범주형 변수 (categorical variable)[[2]](#footnote-3)인 경우입니다. 범주형 변수는 취할 수 있는 값의 수가 제한적이며, 일반적으로 취하는 값이 어떠한 그룹의 이름을 의미하게 됩니다. 성별, 정치 성향, 질병 여부 등이 그러한 예입니다. 성별 변수는 ‘남성’과 ‘여성’이라는 두 개의 값을 취할 수 있고, ‘남성’은 남성들로 구성된 그룹을, ‘여성’은 여성들로 구성된 그룹을 의미합니다. 정치 성향 변수의 경우에는 일반적으로 ‘보수’, ‘중도’, ‘진보’의 세 개의 값을 취할 수 있으며 각 값들은 특정한 정치 성향을 갖고 있는 사람들로 구성된 그룹을 의미합니다.

회귀 문제의 경우는 일반적으로 MSE (mean squared errors)라는 비용함수가 사용되고[[3]](#footnote-4), 분류 문제는 교차 엔트로피 (cross entropy)라고 하는 비용함수가 사용됩니다.

MSE는 아래와 같습니다.

위 식에서 은 학습 데이터에 존재하는 관측치의 수를, 은 학습 데이터에 존재하는 번째 관측치의 실제 종속변수 값을, 그리고 는 우리가 선택한 수학적 모형을 통해 예측되는 번째 관측치가 갖는 종속변수의 예측치가 됩니다. 예를 들어, 우리가 선택한 수학적 모형이 1차 함수이고 학습 데이터에 독립변수가 한 개만 존재한다면 가 됩니다. 는 학습 데이터에 존재하는 번째 관측치가 갖는 독립변수의 값이 됩니다. 실제의 종속변수의 값과 모형을 통해서 예측되는 종속변수의 값의 차이가 작으면 작을수록 비용함수의 값이 작게 됩니다.

종속변수가 취하는 값이 0과 1 두 개인 경우의 교차 엔트로피는 아래와 같이 표현됩니다.

여기서도 은 학습 데이터에 존재하는 관측치의 수를, 은 학습 데이터에 존재하는 번째 관측치의 실제 종속변수 값을 의미합니다. 종속변수가 취할 수 있는 값이 0과 1이기 때문에 이 됩니다. 는 번째 관측치의 종속변수 값이 1일 확률을, 는 번째 관측치의 종속변수 값이 0일 확률을 의미합니다. 분류 문제의 경우는 우리가 선택한 수학적인 모형을 이용해서 와 이 계산됩니다. 종속변수의 실제값이 0인 경우는 의 값이 크게 나올수록, 종속변수의 실제값이 1인 경우는 의 값이 크게 나올수록 비용함수의 값이 작아지게 됩니다.

MSE 또는 교차 엔트로피와 같은 비용함수의 값은 선택되어진 수학적 모형이 갖는 파라미터 값에 의해 달라지게 됩니다. 즉, 비용함수는 파라미터에 대한 함수가 되는 것입니다. 우리는 학습을 통해서 파라미터가 취할 수 있는 값들 중에서 비용함수의 값을 최소화하는 값을 찾게 됩니다. 그리고 이렇게 찾아진 파라미터의 값이 파라미터의 최적값이 되는 것입니다.

예를 들어, 그림 7.4와 같은 비용함수가 있다고 가정해 보겠습니다. 그림을 통해 알 수 있듯이 비용함수의 값은 파라미터인 b가 어떠한 값을 취하느냐에 따라 달라집니다. 그리고 비용함수의 값은 파라미터의 값이 b\*인 경우 제일 작아집니다. b\*이 파라미터의 최적값 (optimal value)가 되는 것입니다. 그리고 b\*이 학습을 통해서 도출되는 값이 됩니다.

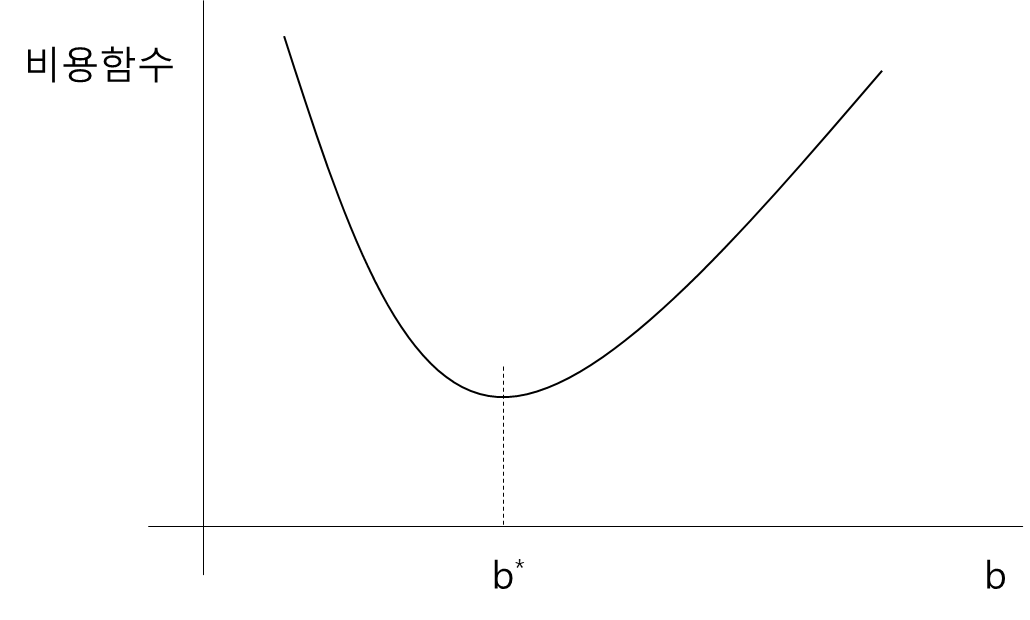


그림 . 비용함수의 예

**파라미터 최적값의 계산**

비용함수의 값을 최소화하는 파라미터의 값이 어떻게 계산되는지를 설명하기 위해서 구체적인 예를 들어보겠습니다. 그림 7.5와 같이 두 개의 관측치로 구성된 학습 데이터가 있다고 가정합니다 (각 관측치는 한 명의 사람을 의미합니다). 독립변수는 직장 경력이고 종속변수는 연봉입니다. 종속변수가 연속형 변수이기 때문에 우리가 풀고자하는 문제의 종류는 회귀 문제가 됩니다. 따라서 MSE 비용함수를 사용합니다.

|  |  |
| --- | --- |
| X (직장 경력) | y (연봉) |
| 1 | 2 |
| 2 | 6 |

그림 . 학습 데이터의 예

위 학습 데이터에 대한 MSE 비용함수는 아래와 같이 표현됩니다 (여기서는 비용함수를 E로 표현하겠습니다).

학습 데이터에 존재하는 X와 y의 관계를 파악하기 위해서 수학적 모형을 선택해야 하는데, 여기서는 설명을 위해 의 간단한 모형을 선택했다고 가정합니다. 우리가 선택한 모형에는 파라미터가 한 개만 존재합니다 (즉, ). 는 우리가 선택한 수학적 모형을 이용해서 계산됩니다. 즉, 가 됩니다. 이를 이용해서 비용함수를 다시 표현하면 아래와 같습니다.

학습 데이터에 존재하는 각 관측치의 X와 y 값을 이용해서 위의 식을 표현하면 아래와 같이 됩니다.

비용함수 E는 파라미터()에 대한 함수인 것을 알 수 있습니다 (즉, 비용함수의 값이 의 값에 따라 달라집니다). 우리는 이 취할 수 있는 여러가지의 값들 중에서 E의 값을 최소화하는 값을 찾아야 합니다. 비용함수 E를 그래프로 나타내면 그림 7.6과 같은데, 여기서 우리는 값을 찾아야 하는 것입니다.

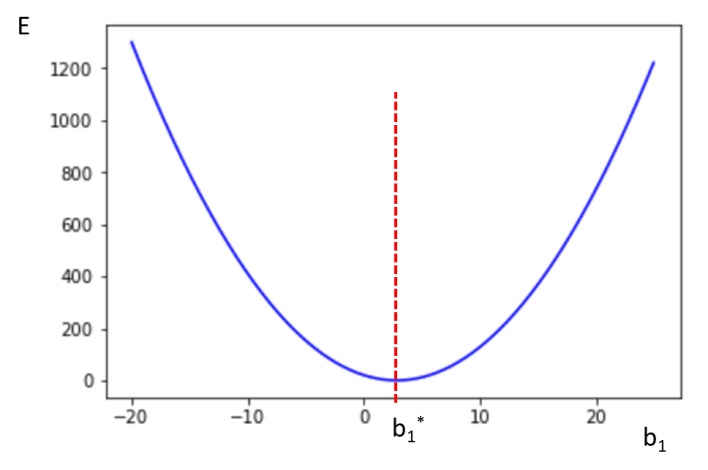


그림 . 비용함수 형태

**비용함수를 최소화하는 파라미터 값 찾기**

그렇다면 비용함수를 최소화하는 파라미터의 값은 어떻게 찾을 수 있을까요? 예를 들어, 그림 7.6에서 은 어떻게 찾을까요?

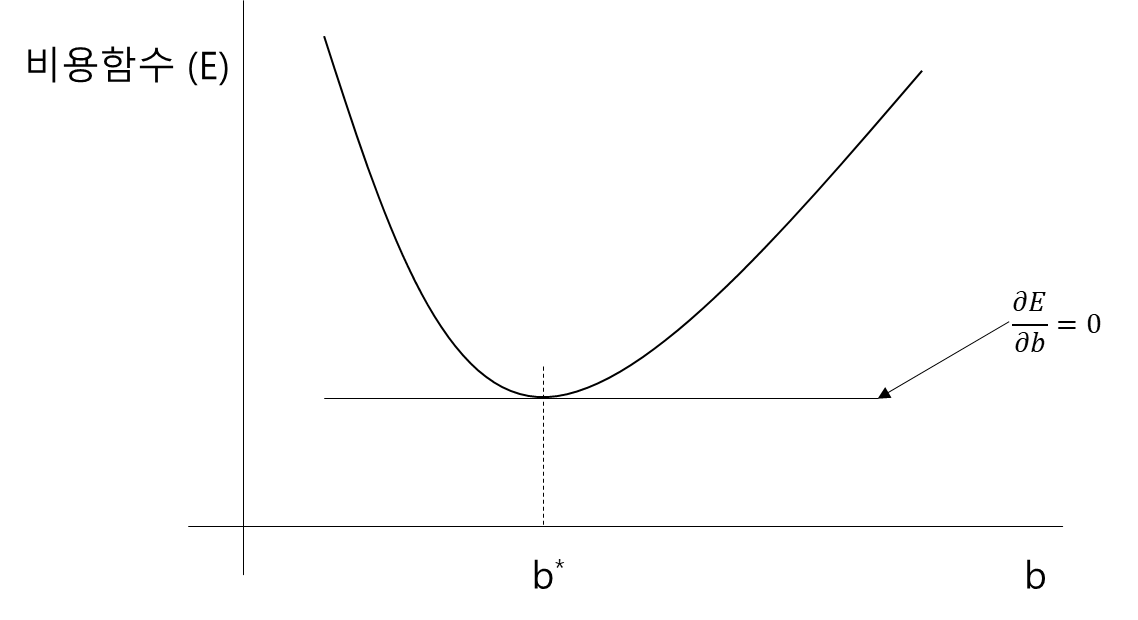
일반적으로 다음 두 가지 방식이 사용됩니다.[[4]](#footnote-5)

① 정규방정식 (Normal equation) 방법

② 경사하강법 (Gradient descent method)

**정규방정식 방법**

정규방정식은 ‘1차 도함수 = 0’인 방정식을 의미합니다. 1차 도함수는 접선의 기울기를 나타내기 때문에 ‘접선의 기울기 = 0’인 방정식이라고 생각할 수 있습니다. 정규방정식 방법은 일반적으로 비용함수가 아래로 볼록한 형태인 경우에 사용합니다.[[5]](#footnote-6) 이는 아래로 볼록한 함수의 경우, 최소가 되는 지점이 한 곳만 존재하며 해당 지점에서는 접선의 기울기가 0이 됩니다 (그림 7.7 참고).

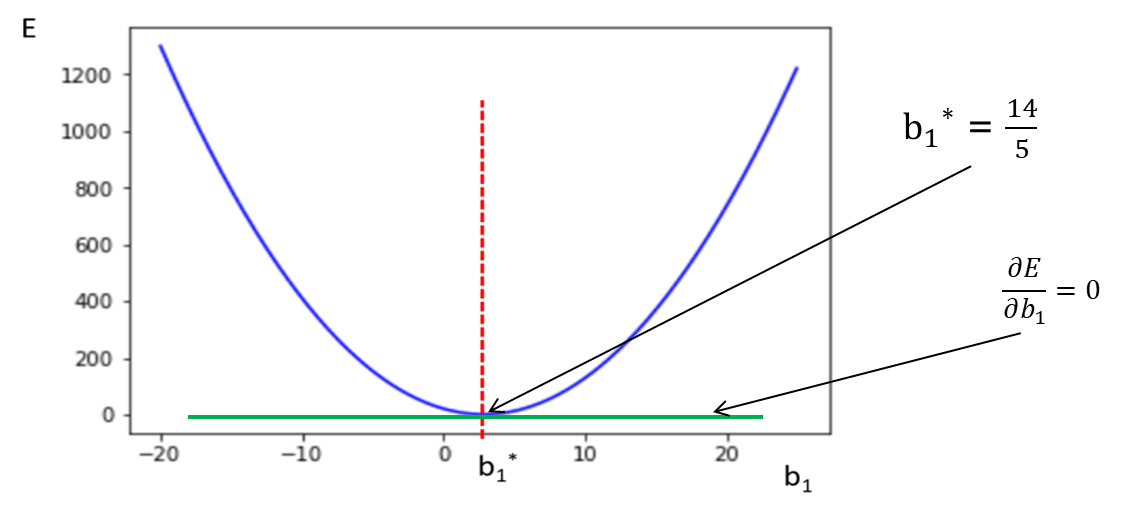


**그림 7.7 볼록함수의 예**

그림 7.7에서 볼 수 있는 것 처럼 접선의 기울기는 비용함수 (E)를 파라미터에 대해서 한 번 미분한 값 (즉, 1차 도함수, )이 됩니다. 볼록 함수에서는 접선의 기울기가 0이 되는 지점이 최소 지점이 되기 때문에 ‘접선의 기울기 = 0’인 방정식을 풀어서 비용함수를 최소화하는 파라미터의 값을 찾을 수 있습니다.

그림 7.5의 학습 데이터에 대한 MSE 비용함수는 아래와 같습니다.

비용함수에 대한 접선의 기울기는 비용함수를 파라미터인 에 대해 한 번 미분한 값, 이 됩니다 (미분 방법에 대해서는 아래 ‘미분 공식’을 참고하세요). 위의 비용함수의 값을 최소로 하는 의 값은 을 만족하는 값이 됩니다. 즉, 가 되는 것입니다. 이를 그림으로 표현하면 아래와 같습니다.



**그림 7.8**

파라미터의 수가 두 개 이상인 경우에는 각 파라미터에 대해서 ‘1차 도함수 = 0’인 방정식을 도출하고 도출된 연립 방정식을 풀어야 합니다. 예를 들어서, 선택된 수학적 모형이 파라미터를 두 개 (예, ) 갖는다라는 경우에는 두 개의 정규 방정식이 도출됩니다.

위 방정식을 동시에 만족하는 의 값을 구하면 됩니다. 이러한 연립 방정식을 풀기 위해서 일반적으로 행렬 연산 방법을 사용합니다.

참고. 미분 공식

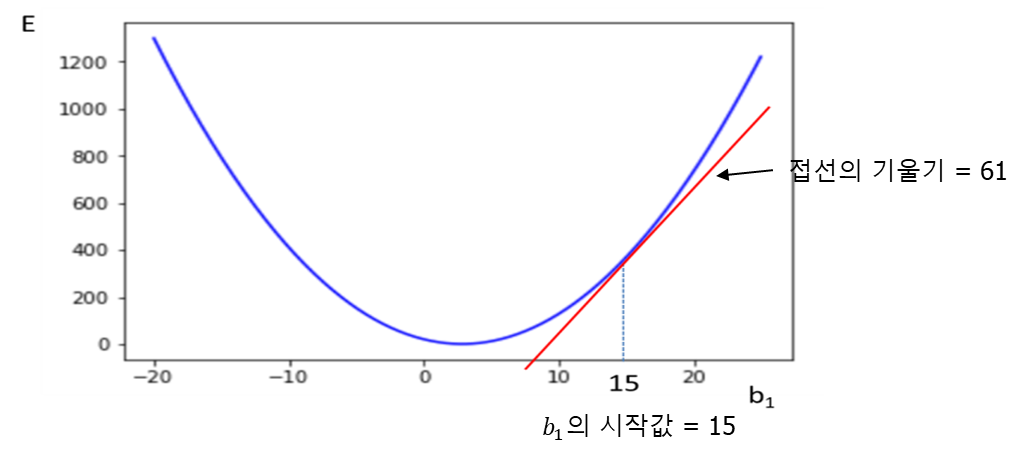
|  |  |  |  |
| --- | --- | --- | --- |
| **함수** |  |  | **예** |
| 다항 함수 |  |  |  |
| 분수 함수 |  |  |  |
| 지수 함수 |  |  |  |
| 로그 함수 |  |  |  |

**경사하강법**

비용함수의 형태가 아래로 볼록한 형태가 아니거나 계산을 해야하는 파라미터의 수가 많은 경우에는 정규방정식 방법을 사용하기가 어렵습니다. 그러한 경우에는 일반적으로 경사하강법이 사용됩니다. 경사하강법은 어떠한 방정식을 풀어서 한 번에 파라미터의 최적값을 찾는 것이 아니라, 파라미터의 값을 순차적으로 업데이트하면서 비용함수를 최소화하는 파라미터의 값을 찾는 방법입니다. 파라미터의 값을 업데이트할 때 사용되는 식은 아래와 같습니다.

는 파라미터 의 현재값이고, 는 새롭게 업데이트된 파라미터의 값입니다. 는 현재의 값 (즉, )에서의 접선의 기울기를 의미합니다. (/에타/)는 학습률 (learning rate)로 한 번에 업데이트 되는 정도를 조절하는 상수입니다.[[6]](#footnote-7)

경사하강법이 어떻게 작동하는지를 설명하기 위해서 앞에서 사용한 비용함수()를 계속해서 사용하겠습니다. 일반적으로 파라미터의 초기값은 랜덤하게 정해집니다. 여기서는 의 초기값이 15로 정해졌다고 가정하겠습니다. 그러한 경우에는 첫 번째 업데이트 단계에서 가 됩니다. 이기 때문에, 가 됩니다. 따라서 첫 번째 업데이트 단계에서의 이 됩니다 (그림 7.9 참고). 이라고 가정하면, 첫 번째 단계에서의 업데이트된 의 값은 다음과 같이 계산됩니다.



**그림 7.9 첫 번째 업데이트 단계**

두 번째 단계에서의 파라미터 현재 값은 이전 단계에서 업데이트된 값 (즉, 14.39)이 됩니다. 따라서, 는 에서의 접선의 기울기가 됩니다. 이는 아래와 같이 계산됩니다.

두 번째 단계에서 새로운 파라미터 값은 아래와 같이 계산됩니다 (두 번째 단계에서의 도 0.01이 됩니다. 이 값은 업데이터 단계에 따라 달라지지 않습니다.).

경사하강법은 이러한 업데이트 과정을 비용함수의 최소로 하는 파라미터 값 (혹은 근사치)을 찾을 때 까지 반복합니다. 일반적으로 정확한 최적값을 찾기 보다는 근사치를 찾습니다. 이는 업데이트 되는 정도에 대해 한계값 (threshold)를 정해, 업데이트 되는 정도가 특정한 값보다 작아질 때 까지 업데이트 과정을 반복하게 됩니다.

**학습의 결과로 도출된 모형을 풀고자하는 문제에 대한 데이터에 적용하기**

앞의 예에서 학습을 통해서 우리가 선택한 수학적 모형 (즉, )이 갖는 파라미터()의 최적값 ()을 도출했습니다. 즉, 학습을 통해 도출된 구체적인 파라미터 값을 갖는 모형은 이 됩니다. 이러한 모형을 풀고자 하는 문제에 대한 데이터에 적용해서 해당 데이터에 존재하는 각 관측치의 종속변수 값을 예측할 수 있습니다. 풀고자 하는 문제에 대한 데이터가 그림 7.10과 같다고 가정합니다. 해당 데이터에는 두 사람 (즉, 두 개의 관측치)에 대한 정보가 저장되어 있습니다. 하지만, 문제에 대한 데이터이기 때문에 학습 데이터와 다르게 각 관측치에 대한 독립변수 (X, 경력) 정보만 존재할 뿐, 종속변수(y, 연봉)에 대한 정보는 존재하지 않습니다.

|  |  |
| --- | --- |
| **관측치** | **X (경력)** |
| 1 | 5 |
| 2 | 15 |

**그림 7.10 문제에 대한 데이터**

우리는 학습의 결과로 도출된 모형을 이용해서 각 관측치의 종속변수 값을 예측해야 합니다. 이를 위해 각 관측치의 독립변수 값을 모형에 대입합니다. 예를 들어, 첫 번째 관측치의 경우는 X=5이기 때문에, 를 얻게 되고, 두 번째 관측치의 경우는 를 얻게 됩니다.

지금까지 설명한 지도학습 알고리즘을 적용하는 순서를 정리하면 아래와 같습니다.

① 학습 데이터와 문제 데이터 준비

② 지도학습 알고리즘 (수학적 모형) 선택

③ 학습 ⇒ 파라미터의 최적값 도출

④ 학습 결과를 문제 데이터에 적용

하지만 일반적으로 과정 ③ 이후 과정 ④를 바로 수행하지 않습니다. 학습 과정을 거친 이후, 학습의 결과도 도출된 모형의 성능을 평가하는 과정을 거치게 됩니다. 즉, 전체적인 지도학습 알고리즘을 적용하는 순서는 아래와 같습니다.

① 학습 데이터와 문제 데이터 준비

② 지도학습 알고리즘 (수학적 모형) 선택

③ 학습 ⇒ 파라미터의 최적값 도출

④ 모형의 성능 평가하기

⑤ 학습 결과를 문제 데이터에 적용

**모형의 성능 평가하기**

학습의 결과로 도출된 구체적인 파라미터 값을 갖는 모형의 성능을 평가하여 모형의 성능이 괜찮고 판단되면 해당 모형을 이용해서 주어진 문제를 풀게 되고, 그렇지 않으면 이전 단계로 돌아가 모형의 성능을 개선하기 위한 작업을 수행합니다. 이러한 작업의 예로는 더 많은 학습 데이터 수집하기, 모형이 갖는 하이퍼파라미터 값 튜닝하기, 다른 모형 사용해 보기 등이 있습니다.

모형의 성능을 평가하기 위해서도 데이터가 필요한데, 평가에 사용되는 데이터 (즉, 평가 데이터)는 다음 두 가지 조건을 만족해야 합니다. 첫 번째로 힌트 정보 뿐만 아니라, 정답 정보도 있어야 합니다. 정답 정보가 존재해야 모형을 통해서 예측된 값이 정답과 얼마나 차이가 나는지를 파악할 수 있습니다. 두 번째 조건은 평가 데이터는 학습에 사용되지 않은 데이터여야 한다는 것입니다. 평가 데이터의 목적은 학습의 결과로 도출된 모형이 학습에 사용되지 않은 데이터에 대해서 어느 정도의 성능을 보이는지를 평가하는 것이기 때문에 평가 데이터는 정답 정보를 포함하되 학습에 사용되지 않은 데이터여야 합니다. 따라서 정답과 힌트 정보가 있는 데이터를 모두 학습 데이터로 사용하지 않고, 이중 일부를 학습 데이터로, 나머지 일부를 평가 데이터로 사용하게 됩니다. 보통 학습 데이터와 평가 데이터의 비율은 7:3 또는 8:2 정도가 됩니다. 하지만, 정답 데이터의 양이 더 많은 경우에는 학습 데이터의 비율을 더 크게 하는 것이 일반적입니다.

모형의 성능 평가의 목적으로 사용되는 지표에는 여러가지가 있습니다. 그리고 이러한 지표는 지도학습 알고리즘을 이용해서 풀고자 하는 문제의 종류가 회귀 문제인지 분류 문제인지에 따라서 달라집니다. 회귀문제에서는 RMSE, RMSLE, R2 등이 사용되고, 분류 문제의 경우는 정확도 (accuracy), 재현율 (recall), 정밀도 (precision), F1, AUC, log loss 등의 지표가 사용됩니다.

여기서는 회귀 문제에서 사용되는 평가 지표들에 대해서 간단하게 살펴보겠습니다. 분류 문제에서 사용되는 지표에 대해서는 텍스트 분류 부분 (12 장)에서 자세하게 다룹니다.

RMSE (Root Mean Squared Errors)는 비용함수인 MSE에 루트를 취한 값입니다. 식은 아래와 같습니다.

RMSLE (Root Mean Squared Log Errors)는 아래와 같습니다.

RMSE와 RMSLE 모두 그 값이 작을수록 모형의 성능이 좋다는 것을 의미합니다.

R2은 아래와 같이 표현됩니다.

위 식에서 는 학습 데이터에 존재하는 종속변수의 평균을 의미합니다. 분모는 학습 데이터에 대해서 종속변수가 갖는 전체의 흩어진 정도를 의미하고, 분자는 모형에 의해 설명되는 흩어진 정도를 의미합니다. 즉, 은 종속변수가 갖는 전체의 흩어진 정도 중에서 모형에 설명되는 정도를 의미합니다. 은 0 – 1 값을 취하고 1에 가까울수록 모형의 설명력이 좋다는 것을 의미합니다.

### 지도학습에서의 과적합 (overfitting) 문제

로지스틱 회귀 모형은 지도학습 알고리즘의 한 종류라고 말했습니다. 지도학습의 경우에 가장 유의해야 하는 문제 중 하나가 과적합 문제 (overfitting problem)입니다. 과적합이라고 하는 것은 우리가 사용하는 모형이 학습 데이터에 존재하는 독립변수와 종속변수의 관계는 아주 잘 설명하는데, 그에 비해 새로운 데이터 (unseen data)에 대해서는 종속변수의 값을 잘 예측하지 못하는 것을 의미합니다. 우리가 지도학습 알고리즘을 이용해서 궁극적으로 해결하고자 하는 것은 새로운 데이터에 존재하는 종속변수를 정확하게 예측하고자 하는 것이기 때문에 이러한 과적합 문제는 우리의 모형의 성능이 새로운 데이터에 대해서 별로 좋지 못하다는 것을 의미합니다. 따라서 지도학습에서는 모형이 갖는 과적합 문제를 해결하는 것이 무척이나 중요합니다. 과적합 문제를 해결하는 방법에는 여러가지가 있습니다. 그 중에서 가장 일반적으로 많이 사용되는 방법이 regularization 방법입니다. Regularization 방법은 학습의 결과로 도출되는 파라미터의 값을 줄여 각 독립변수가 종속변수에 미치는 영향을 줄임으로써 과적합 문제를 해결합니다. 즉, 우리의 모형이 학습데이터 민감하게 반응하는 정도를 줄여주는 것입니다. 우리의 모형이 학습 데이터에 대해서 민감하게 반응하게 되면 그 만큼 학습 데이터는 잘 설명하는 반면에 새로운 데이터에 대한 설명력은 낮아질 확률이 커지게 됩니다. 따라서 파라미터의 값을 줄여줌으로써 우리의 모형이 학습데이터에 민감하게 반응하는 정도를 줄여줄 수 있습니다. 설명을 위해서 다음과 같은 간단한 선형회귀모형이 있다라고 가정하겠습니다.

독립변수를 2개 갖는 모형입니다. 그리고 우리가 학습을 통해서 해야하는 것은 학습 데이터에 존재하는 독립변수들과 종속변수의 관계를 가장 잘 설명하는 파라미터들 (즉, )의 값을 찾아야 하는 것입니다. 이러한 파라미터들 중에서 은 X1이 종속변수에 영향을 미치는 정도 (혹은 X1에 의해서 y값이 달라지는 정도), 는 X2가 종속변수에 영향을 미치는 정도를 의미합니다. 즉, (또는 )의 값이 크면 클수록 종속변수의 값이 X1 (또는 X2)에 민감하게 반응한다는 것을 의미합니다. 즉, 그 만큼 우리의 모형이 학습 데이터에 민감하다는 것을 의미하는 것이지요. 예를 들어 보겠습니다. 의 값이 10인 경우와 1인 경우를 비교해 보도록 하겠습니다. =10 인 경우에는 X1의 값이 1만큼 증가할 때 y의 값이 10만큼 증가한다는 것을 의미합니다. 그리고 =1인 경우에는 X1의 값이 똑같이 1만큼 증가할 때 y의 값이 1만큼 증가한다는 것을 의미합니다. 즉, 의 (절대)값이 작을 때 종속변수가 모형에 덜 민감하게 반응합니다. 이렇게 하면 과적합 문제를 어느 정도 줄일 수 있습니다.

로지스틱 회귀에서는 y=1일 확률 (혹은 y=0일 확률)이 독립변수에 덜 민감하게 반응한다는 것을 의미합니다. 감성분석에서는 특정 단어의 출현 횟수에 영화평이 긍정일 확률 (또는 부정일 확률)이 덜 민감하게 반응한다는 것을 의미합니다.

모형이 갖는 파라미터의 절대값을 줄이는 방법 중에 하나가 regularization입니다. regularization 모형이 갖는 비용함수에 파라미터와 관련된 추가적인 항을 더해 줌으로써 파라미터의 절대값을 줄여주는 방법입니다. Regulization은 어떠한 형태의 항을 더해주느냐에 따라서 그게 두가지 방법으로 구분됩니다. 하나는 L1 방법이고 다른 하나는 L2 방법입니다. L1 방법은 Lasso 방법으로, L2 방법은 Ridge 방법으로도 표현됩니다. L1 방법과 L2 방법을 이해하기 위해서는 벡터 관련 섹션에서 다뤘던 Lp-Norm 개념에 대해서 알아야 합니다. k개의 원소를 갖는 어떠한 벡터에 대해서 Lp-Norm은 다음과 같이 정의 됩니다. 이를 위해서 k개의 원소를 갖는 **x**라는 벡터가 있다고 가정하겠습니다. 즉, 에 대한 Lp-Norm은 다음과 같이 정의됩니다.

p=1인 경우를 L1 norm이라고 하고 p=2인 경우를 L2 norm이라고 합니다. 즉, L1 norm은

L2 norm은 아래와 같은 형태가 됩니다.

L1 regulization 방법은 이러한 L1 norm을 사용한 것이고, L2 regularization 방법은 L2 norm을 이용한 것입니다.

우리의 (로지스틱 회귀) 모형이 k개의 파라미터를 갖고, 해당 파라미터는 다음과 같이 벡터로 표현된다고 가정합니다.

그리고 해당 모형의 비용함수가 다음과 같이 파라미터의 함수로 표현된다고 가정하겠습니다.

위의 비용함수는 앞에서 살펴본 교차 엔트로피 비용함수가 되는 것입니다. 우리는 이러한 비용함수의 값을 최소화하는 파라미터의 값 즉, **w** 벡터의 원소 값들을 학습을 통해서 얻게 됩니다. 하지만, 위의 비용함수만을 이용하면 과적합 문제가 발생할 확률이 높아지게 됩니다. 과적합 문제를 줄이기 위해서 계산되어 지는 파라미터의 값들을 줄일 필요가 있는데, 이러한 목적으로 비용함수에 파라미터와 관련된 어떠한 항을 추가하여 새로운 형태의 비용함수를 만들어 줍니다. L1 regularization은 아래와 같은 L1-norm을 더해 줍니다.

L1 regularization에서의 새로운 비용함수: +

여기서 이며, /람다/는 penalty strength라고 불리는 상수입니다 (이는 사용자가 결정하는 하이퍼파라미터입니다). 의 값을 크게할수록 파라미터의 값이 더 많이 줄어드는 효과가 있습니다.

L2 regularization은 아래와 같은 L2-norm 형태를 그대로 더해 주는 것이 아니라, 아래와 같이 L2 norm의 제곱형태를 더해주게 됩니다.

L2 regulization에서의 새로운 비용함수: +

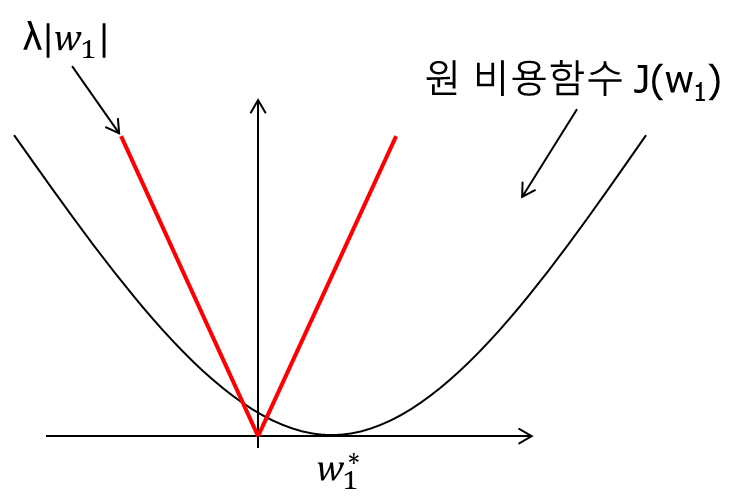
여기서 은 가 됩니다.

**L1 regularization과 L2 regularization의 차이**

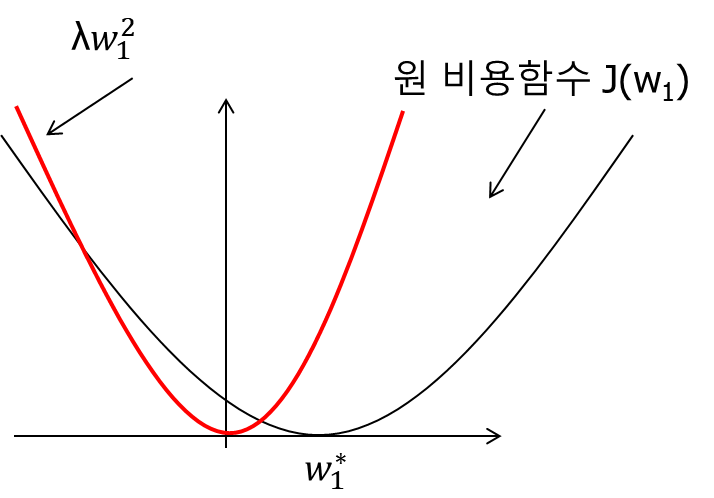
L1과 L2 방법 모두 학습의 결과로 얻어지는 파라미터의 (절대)값을 줄여주는 효과가 있습니다. 하지만, L1는 파라미터의 값이 줄어서 0까지 줄어들 수 있습니다. 그에 반해서 L2는 0까지 줄어들지는 않습니다. L1 방식은 feature selection의 역할을 한다고 볼 수 있습니다. 즉, 상대적으로 중요한 역할을 하는 feature (즉, 독립변수)만을 남겨놓고, 덜 중요한 feature들은 모형에서 제외하는 효과가 있는 것입니다.[[7]](#footnote-8)

좀 더 직관적으로 설명하기 위해서 파라미터가 하나만 있는 경우에 대해서 L1과 L2의 차이를 보도록 하겠습니다.

L1의 경우는 아래와 같이 w1에 대해서 절대값 함수가 더해진다고 생각할 수 있습니다. 원 비용함수의 값을 최소화하는 w1의 값은 이 됩니다. 우리가 L1 항을 추가하지 않고 원 비용함수를 가지고 학습을 시킨다면 결과로 값을 얻게 되는 것입니다. 하지만, 원 비용함수에 절대값 형태의 새로운 함수가 더해진 새로운 비용함수의 값을 최소화하는 w1 값은 보다 작어지게 됩니다. 즉, 해당 독립변수의 영향이 그 만큼 줄어드는 것입니다. 그리고 절대값 함수가 더해지는 경우에는 새로운 비용함수의 최솟값이 w1=0 인 지점에서 발생할 수 있는 것입니다. 그러한 경우는 해당 독립변수가 모형에서 제외되는 효과가 있습니다.



이에 반해, L2의 경우는 아래와 같이 w1에 대한 2차함수가 원 비용함수에 더해 집니다.



이렇게 되면 새로운 비용함수를 최소화하는 w1의 값이 기존의 값보다 줄어들기는 하지만, 그 지점이 w1=0 되지는 않습니다.

## 벡터 (Vector)

기계학습 알고리즘은 기본적으로 데이터에 존재하는 각 관측치를 벡터로 변환하여 처리합니다. 따라서 기계학습 알고리즘을 보다 잘 이해하기 위해서는 벡터의 기본적인 내용에 대해서 알고 있는 것이 필요합니다.

### 벡터란?

벡터란 간단하게 표현하면 여러 개의 숫자들을 일렬로 배열한 것이라고 할 수 있습니다. 숫자들을 가로로 배열한 경우에는 횡벡터 (row vector), 세로로 배열한 경우에는 종벡터 (column vector)라고 합니다. 다음과 같이 표현할 수 있습니다. 일반적으로 벡터라고 하면 종벡터를 의미합니다.

(횡벡터)

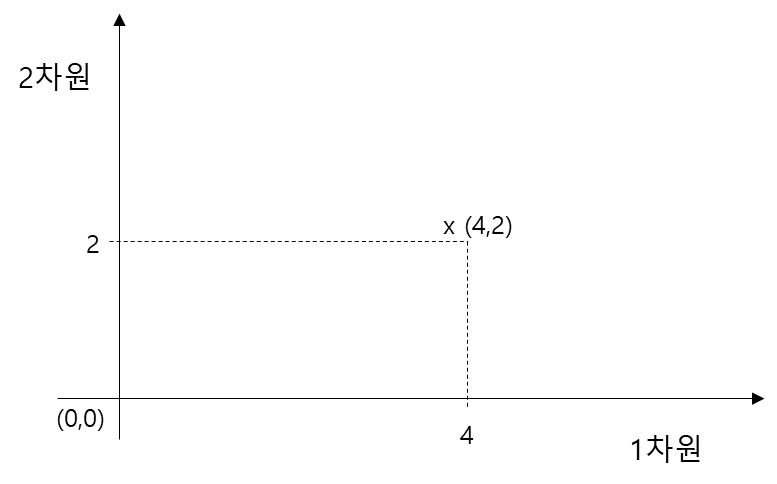
(종벡터)

표현의 차이만 있지 둘은 같은 벡터입니다. 벡터를 구성하는 각 숫자를 원소 (element)라고 합니다. 위의 벡터 a는 세 개의 숫자로 구성되어 있기 때문에 원소의 수가 3인 벡터가 됩니다.

기계학습 혹은 데이터 분석에서의 벡터가 갖는 역할 혹은 의미를 잘 이해하기 위해서는 벡터의 공간적인 의미 (혹은 기하학적 의미)를 이해하는 것이 필요합니다. 즉, 공간 상에서 벡터가 갖는 의미를 알아야 합니다. 벡터는 공간 (space)상에서 어떠한 의미를 가질까요?

벡터는 공간 상에 존재하는 하나의 점 (point)을 의미합니다. 공간은 차원(dimension)을 갖습니다. 1차원 공간, 2차원 공간, 3차원 공간이 그러한 예입니다. 그러면 벡터가 존재하는 공간의 차원은 어떻게 결정이 될까요? 이는 벡터가 갖는 원소(element)의 수에 의해서 결정이 됩니다. 즉, 벡터가 존재하는 공간의 차원은 벡터가 갖는 원소의 수와 동일합니다. 예를 들어, 벡터 는 원소의 수가 세 개이므로 3차원 공간에 존재하는 하나의 점이 됩니다.

좀 더 구체적인 설명을 위해서 여러분이 친숙한 2차원 공간에 존재하는 벡터를 예로 들어 보겠습니다. x = (4,2) 라고 하는 벡터가 있다고 가정하겠습니다. 이 벡터는 2차원 공간에 존재하는 하나의 점입니다. 왜냐하면, 원소의 수가 2개이기 때문입니다. 이를 2차원 공간상에 표현하면 그림 7.11과 같습니다. 가로축은 1차원 축이되고 세로 축은 2차원 축이 됩니다. 공간을 표현하는 축은 서로 수직입니다. x = (4,2)에서 4는 1차원 축에 대한 값이 되고, 2는 2차원 축에 대한 값이 됩니다.



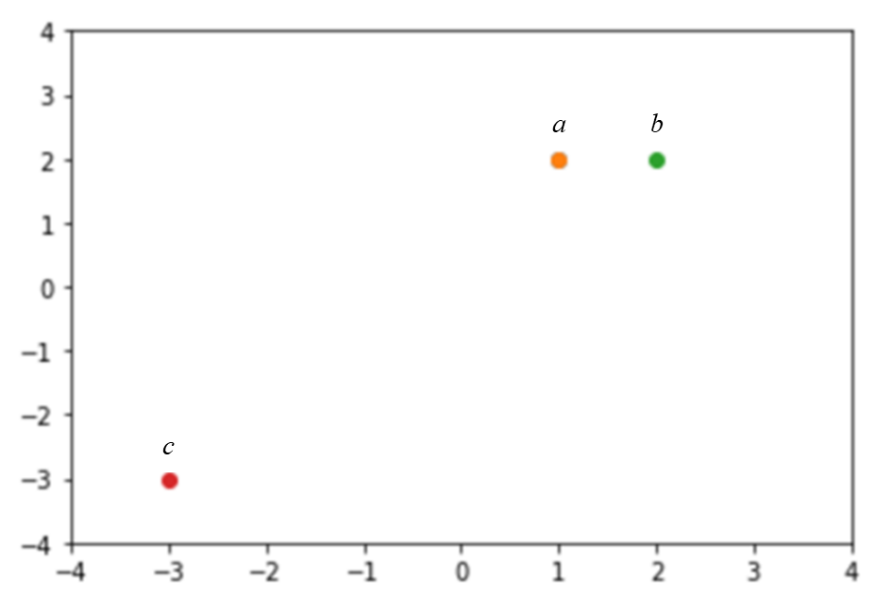
**그림 7.11 2 차원 공간에 존재하는 x (4,2)**

### 공간 상에서 벡터의 위치

벡터는 공간 상에서 특정한 위치를 갖습니다. 그리고 벡터의 위치는 그림 7.11에서 볼 수 있는 것 처럼 벡터가 가지고 있는 각 원소의 값에 의해서 결정이 됩니다. 예를 들어 벡터 x (4,2)는 1차원 축을 기준으로 원점으로 4만큼 떨어져 있고, 2차원 축을 기준으로 원점으로부터 2만큼 떨어진 위치에 존재합니다. 그렇다면, 벡터가 갖고 있는 원소의 값은 무엇을 의미할까요? 벡터의 고유한 특성을 의미합니다. 즉, 벡터의 고유한 특성이 벡터가 갖는 구체적인 원소값에 의해서 표현됩니다. 결국, 이는 **벡터의 위치가 벡터의 고유한 특성을 나타낸다**는 것을 의미합니다. 공간 상에서의 벡터의 위치가 벡터의 고유한 특성을 나타내기 때문에 서로 다른 두 벡터들에 대해서 두 벡터의 위치가 얼마나 가까운지 혹은 먼지가 두 벡터의 고유한 특성의 유사도를 의미합니다. 위치가 가까울수록 고유한 특성의 유사도가 큰 것입니다.

이 번에는 다음과 같은 서로 다른 세 개의 벡터를 가정해 보겠습니다.

모두 원소의 수가 두 개인 벡터로 2차원 공간 상의 점으로 표현됩니다 (그림 7.12 참고).



**그림 7.12 의 위치**

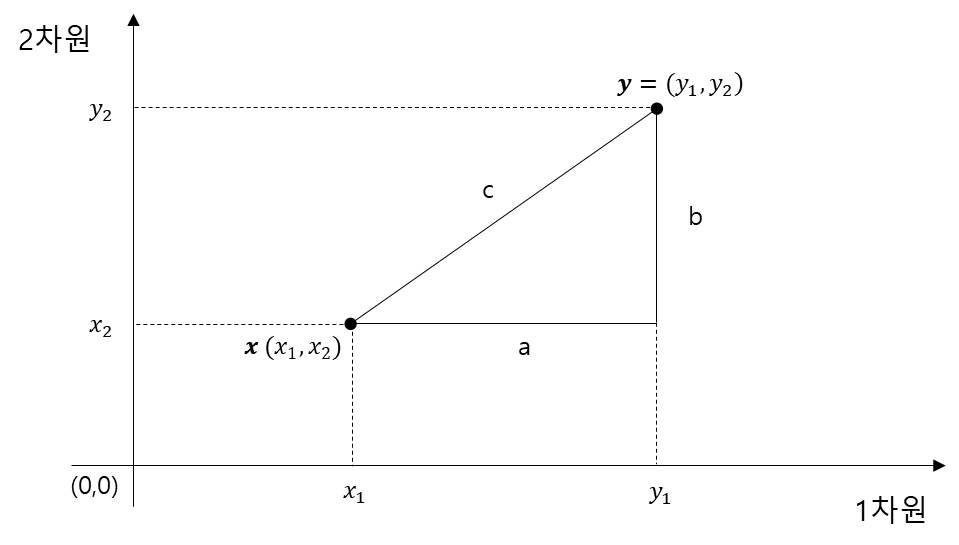
공간 상에서의 위치가 가까울수록 벡터들의 고유한 특성이 더 유사한 특성을 갖는다는 것을 의미하기 때문에 그림 7.12에서 볼 수 있듯이 벡터 은 벡터 보다 벡터 와 더 유사한 특성을 갖는다는 것을 알 수 있습니다.

그렇다면 두 벡터의 위치가 가깝고 먼 정도를 어떻게 수치적으로 표현할까요? 이는 두 벡터 사이의 거리를 이용합니다. 거리가 짧을수록 위치가 가깝다는 것을 (즉, 유사도가 더 크다는 것을), 거리가 길수록 위치가 멀다는 것을 (즉, 유사도가 작다는 것을) 의미합니다.

### 유클리디안 거리 (Euclidean distance)

두 벡터 간의 거리를 계산하는 방법에 대해서 살펴보도록 하겠습니다. 가장 일반적으로 많이 사용되는 방법이 유클리디안 거리 입니다. 유클리디안 거리는 두 벡터 사이의 직선의 거리를 나타냅니다. n 차원 공간에 존재하는 두 벡터 와 사이의 유클리디안 거리는 아래와 같이 정의됩니다.

유클리디안 거리가 어떻게 계산되는지 설명하기 위해서 2차원 공간에 존재하는 두 벡터 와 를 예로 들어 보겠습니다 (그림 7.13 참고). 위 정의에 따라 두 벡터 와 의 유클리디안 거리는 가 됩니다. 이는 그림 7.13에서 c의 길이와 같습니다. 그림 7.13에서 a, b, c 간의 관계는 피타고라스 정리에 의해 입니다. 여기서 a는 직각 삼각형의 밑변의 길이를 의미하며 이는 이고, b는 직각 삼각형에서의 높이를 의미하고 이는 가 됩니다. 따라서 c 즉, 와 의 유클리디안 거리는 가 되는 것입니다.



**그림 7.13 유클리디안 거리 계산의 예**

앞에서 다룬 세 벡터 에 대해서 와 사이의 유클리디안 거리와 와 사이의 유클리디안 거리 계산해 보겠습니다. 이는 아래와 같습니다.

,

벡터 와 사이의 거리는 1이고 와 사이의 거리는 6.403이 됩니다. 공간에서의 벡터간의 거리는 벡터간의 유사도를 의미하기 때문에, 위의 결과를 토대로 는 보다 하고 대략 6.4배 정도 더 유사하다는 것을 알 수 있습니다.

### 파이썬에서 벡터 다루기[[8]](#footnote-9)

#### 벡터 생성하기

파이썬에서는 일반적으로 numpy 모듈을 이용해서 벡터를 다룹니다.[[9]](#footnote-10) numpy를 사용하기 위해서 다음과 같이 임포트 합니다.

import numpy as np

numpy를 이용해서 벡터를 생성할 때는 numpy에서 제공되는 array() 함수를 사용합니다. 리스트데이터를 array() 함수의 인자로 제공하여 벡터를 생성할 수 있습니다.[[10]](#footnote-11) 예를 들어, a = (1, 2) 벡터를 numpy를 이용하여 생성하기 위해서는 아래와 같이 입력합니다.

a = np.array([1, 2])

위와 같이 벡터를 생성한 후 shape이라는 키워드를 적용해서 생성된 어레이 데이터의 형태를 확인할 수 있습니다.

a.shape

결과는 아래와 같습니다. 첫 번째 원소는 행의 갯수를 두 번째 원소는 열의 갯수를 표현합니다. numpy에서의 벡터는 열의 갯수가 표시되지 않습니다. 아래 결과는 원소가 2개인 벡터라는 것을 의미합니다.

(2,)

벡터 a의 원소 갯수는 len() 함수를 사용해서도 확인할 수 있습니다.

len(a)

2

#### 벡터의 길이 계산하기

파이썬에서는 numpy의 하위 모듈인 linalg[[11]](#footnote-12)에서 제공하는 norm() 함수[[12]](#footnote-13)를 이용해서 벡터의 길이 를 계산할 수 있습니다. 여기서의 벡터의 길이는 원점으로부터의 유클리디안 거리가 됩니다. 예를 들어서, 벡터 의 길이는 원점 과 사이의 유클리디안 거리 즉, 가 됩니다. 이를 norm() 함수를 사용하면 다음과 같이 계산할 수 있습니다. 길이를 구하고자 하는 벡터를 norm() 함수의 인자로 입력합니다.

np.linalg.norm(a)

위 함수가 반환하는 값은 에 해당하는2.23606797749979 입니다.

#### norm() 함수를 이용해 유클리디안 거리 계산하기

norm() 함수를 이용해서 두 벡터 사이의 유클리디안 거리를 계산해 보도록 하겠습니다. 이를 위해서 먼저 두 벡터의 합과 차가 어떻게 계산되는지 간단히 살펴보겠습니다. 두 벡터의 합은 같은 자리에 있는 원소끼리의 합을 의미하고 두 벡터의 차는 같은 자리에 있는 원소 간의 빼기를 의미합니다. 예를 들어, 두 벡터 와 에 대해서 가 되고 가 됩니다. 두 벡터의 합과 차도 또 다른 하나의 벡터가 되는 것을 알 수 있습니다. 를 로 표현하도록 하겠습니다. 그러면 의 길이 (즉, 원점으로부터의 유클리디안 거리는 가 됩니다. 즉, )의 길이는 두 벡터 와 사이의 유클리디안이 됩니다. 따라서 norm() 함수를 이용해서 벡터를 계산함으로써 간단히 와 의 사이의 유클리디안을 계산할 수 있습니다. norm()와 같이 입력하면 됩니다. 예를 들어, 위에서 살펴본 에 대해서 벡터 와 벡터 사이의 유클리디안 거리는 다음과 같이 계산합니다.

np.linalg.norm(b-a)

화면에 1의 결과가 출력됩니다.

마찬가지로 와 사이의 유클리디안 거리는 아래와 같이 계산합니다.

np.linalg.norm(c-a)

6.4031242374328485

### 벡터의 길이, norm

선형대수에서는 벡터의 길이를 norm이라고 표현합니다. 여러 종류의 norm이 존재하지만, 일반적으로 많이 사용되는 norm이 Lp-norm(/엘피놈/)입니다. 벡터 에 대해 Lp-norm은 다음과 같이 정의됩니다.

데이터 분석에서 가장 Lp-norm은 p의 값이 1인 L1-norm과 2인 L2-norm입니다. L1-norm은 다음과 같이 표현됩니다.

L2-norm은 아래와 같습니다.[[13]](#footnote-14)

두 벡터와 의 차로 표현되는 벡터에 대한 Lp-norm은 아래와 같이 정의 됩니다.

벡터 의 Lp-norm은 두 벡터 와 사이의 Lp-norm으로 표현되는 거리를 의미합니다.

에 대한 L1-norm은 다음과 같이 표현됩니다.

이는 벡터 와 사이의 L1-norm 거리이고, 이를 시티블록 (city-block) 거리 혹은 맨하탄 거리 (Manhattan distance)라고도 합니다. 시트블록 거리는 파이썬에서 제공되는 Scipy 모듈의 하위 모듈인 distance 모듈에서 제공되는 cityblock() 함수를 사용합니다.[[14]](#footnote-15) 앞에서 살펴본 두 벡터 사이의 시티블록 거리를 계산해 보겠습니다. 두 벡터를 cityblock() 함수의 인자로 입력합니다.

import scipy.spatial.distance as dst # distance 모듈 임포트 하기

dst.cityblock(a,b) # |1-2| + |2-2|와 같으므로 1을 반환합니다.

1

두 벡터 사이의 L1-norm 거리는 numpy의 linalg 모듈에서 제공되는 norm() 함수를 이용해서 계산할 수도 있습니다. 아래와 같이 코딩합니다. 이번에는 와 사이의 L1-norm 거리를 계산해 보겠습니다. L1-norm 거리를 계산하기 위해서는 두 번째 인자로 1의 값을 입력합니다.

np.linalg.norm(c-a, 1) # |-3-1|+|-3-2|와 같으므로 결과는 9입니다.

9.0

에 대한 L2-norm은 아래와 같이 표현됩니다.

위의 식에서 알 수 있듯이 두 벡터의 차로 표현되는 벡터의 L2-norm은 두 벡터 사이의 유클리디안 거리와 같습니다. 앞에서는 numpy 모듈에서 제공되는 norm() 함수를 이용해서 계산했었습니다. norm() 함수의 두 번째 인자의 기본값은 2라고 생각할 수 있습니다. 즉, 두 번째 인자의 값을 입력하지 않으면 두 벡터 사이의 L2-norm 거리가 계산됩니다. 유클리디안 거리는 Scipy 모듈의 하위 모듈인 distance 모듈에서 제공되는 euclidean() 함수를 이용해서 계산할 수도 있습니다 (아래 코드 참고).

dst.euclidean(a,c)

6.4031242374328485

### 그 외 자주 사용되는 거리 지표

지금까지 두 벡터 사이의 거리 지표로 유클리디안 거리와 시티블록 거리를 살펴 봤습니다. 그 외 자주 사용되는 거리 계산 방식으로는 자카드(Jaccard) 거리[[15]](#footnote-16)와 해밍(Hamming) 거리[[16]](#footnote-17)가 있습니다. 두 거리 지표 모두 벡터의 원소가 취하는 값이 0 또는 1인 경우에 유용하게 사용할 수 있습니다.

#### 자카드 (Jaccard) 거리

0과 1의 값을 원소로 갖는 두 벡터에 대해서 자카드 거리는 다음과 같이 정의됩니다.

는 같은 자리에 있는 원소들 중에서 첫 번째 벡터의 원소값은 1이고 두 번째 벡터의 원소값은 0인 원소 자리의 수가 됩니다. 예를 들어, 두 벡터 가 있다고 가정하겠습니다. 두 벡터에 대해서 의 값은 0이 됩니다 (즉, 같은 자리에 있는 원소들 중에서 첫 번째 벡터의 원소값이 1이고 두 번째 벡터의 원소값이 0인 원소 자리는 하나도 없습니다). 은 같은 자리에 있는 원소들 중에서 첫 번째 벡터의 원소값은 0이고 두 번째 벡터의 원소값은 1인 원소 자리의 수가 됩니다. 두 벡터 와 에 대해서 값은 1이 됩니다. 두 번째 자리의 원소들을 보면벡터 의 두 번째 원소의 값은 0이고, 벡터 의 두 번째 원소의 값은 1입니다. 그리고 같은 원리로 이 됩니다. 따라서 두 벡터 와 의 자카드 거리는 ½ = 0.5가 됩니다. 자카드 거리는 Scipy의 distance 모듈에서 제공되는 jaccard() 함수를 사용해서 계산할 수 있습니다 (아래 참고).

d = np.array([1,0,0])

e = np.array([1,1,0])

dst.jaccard(d,e)

0.5

위의 공식에서 알 수 있듯이 같은 자리에 있는 원소의 값이 다를 수록 두 벡터 사이의 자카드 거리의 값이 커집니다.

#### 해밍 (Hamming) 거리

0과 1의 값을 취하는 원소들을 갖는 두 벡터에 대해서 해밍 거리는 다음과 같이 정의됩니다.

n은 각 벡터가 갖는 원소의 수를 의미합니다. 과 은 자카드 공식에서와 동일한 의미를 갖습니다. 두 벡터 사이의 해밍 거리는 1/3이 됩니다. 해밍 거리는 Scipy의 distance 모듈에서 제공되는 hamming() 함수를 이용해서 계산합니다.

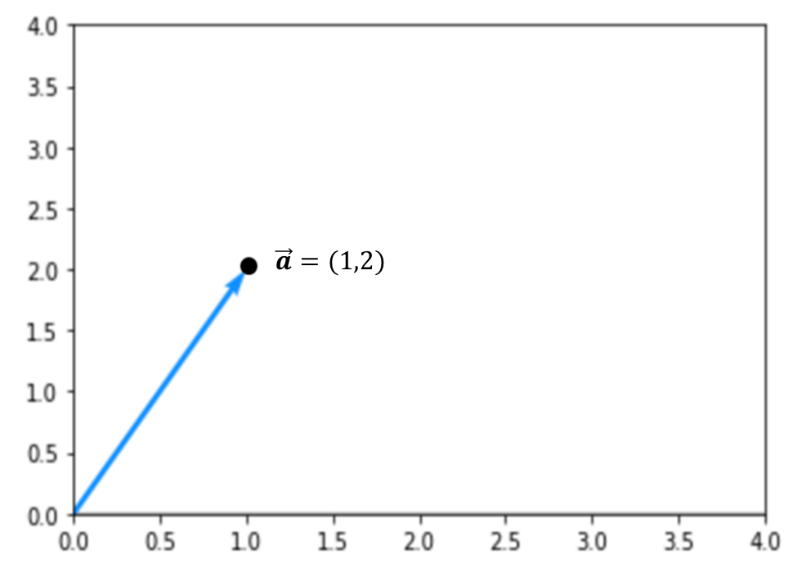
dst.hamming(d,e)

0.3333333333333333

### 코사인 유사도 (Cosine similarity)

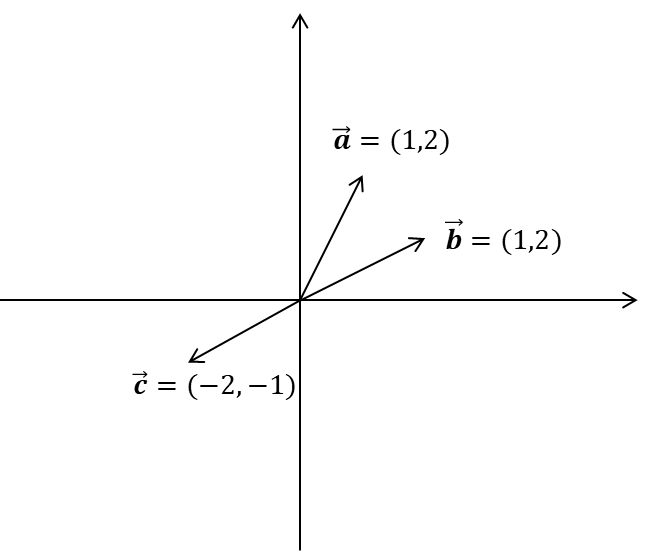
#### 방향을 기준으로한 벡터 간의 유사도

지금까지 배운 벡터 사이의 유사도를 계산하는 방법은 주로 벡터의 위치를 기반으로 한 것들이었습니다. 즉, 가까이 위치할 수록 더 유사하다는 개념을 기반으로한 방법들이었습니다. 벡터 간의 유사도를 측정하는 또 다른 방법으로 코사인 유사도가 있습니다. 이는 거리가 아니라 벡터 간 방향성의 유사도를 기반으로한 방법입니다. 벡터는 방향성이 중요하기도 한데, 벡터의 방향성은 원점을 기점으로 결정됩니다. 예를 들어, 벡터 의 방향은 아래와 같이 표현될 수 있습니다. 방향이 중요한 경우에는 와 같이 화살표 위첨자를 이용하여 표현합니다.



**그림 7.14 벡터 의 방향**

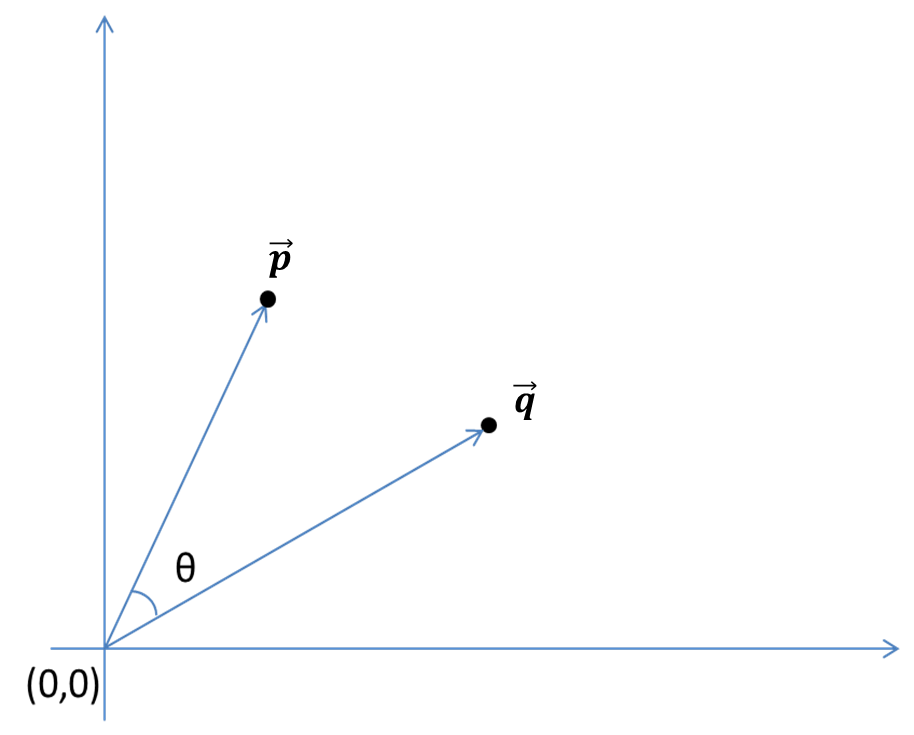
공간에서 벡터가 갖는 방향 또한 벡터가 갖는 원소의 값에 의해서 결정이 됩니다. 다시 한번 강조하지만, **원소의 값은 벡터의 고유한 특성을 나타냅니다**. 즉, 벡터의 방향 또한 벡터의 고유한 특성을 반영하는 것입니다. 이는 방향이 비슷할수록 벡터 간의 고유한 특성이 크다는 것을 의미합니다. 그림 7.15와 같이 2차원 공간에 존재하는 서로 다른 세 개의 벡터가 있다고 가정하겠습니다.



**그림 7.15 방향이 다른 세 벡터의 예**

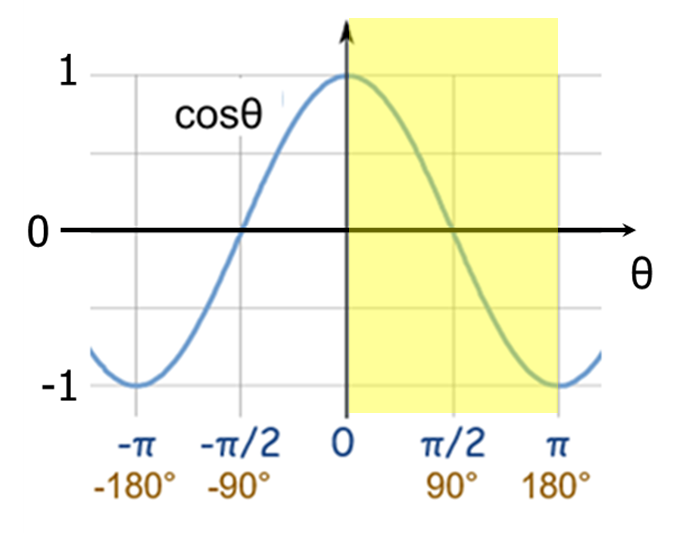
방향을 기준으로 했을 때 벡터 는 벡터 와 중에서 어떤 벡터와 더 유사할까요? 그림에서 볼 수 있듯이 방향이 더 유사한 벡터 와 더 유사합니다.

그렇다면 방향을 기준으로 한 벡터 간의 유사도를 수치로 표현할 수 있는 방법이 없을까요? 바로 그 때 사용할 수 있는 방법이 코사인 유사도 입니다. 코사인 유사도를 설명하기 위해서 그림 7.16과 같은 두 벡터가 있다고 가정합니다.



**그림 7.16 두 벡터 사이의 사이각 (θ)**

두 벡터 사이의 사이각을 θ라고 표현하겠습니다. 사이각 θ는 두 벡터의 방향성 유사도와 어떠한 관계가 있을까요? 두 벡터의 방향성 유사도는 θ의 값에 따라서 달라집니다. 사이각(θ)의 값이 0˚ 에 가까울수록 방향이 유사해지고, 그 값이 180˚에 가까울수록 방향이 달라지게 됩니다. θ = 0˚인 경우 두 벡터는 동일한 방향을 갖고, 방향을 기준으로 했을 때 두 벡터의 유사도가 최대가 됩니다. 사이각이 180˚인 경우에는 방향이 정반대가 되어 방향을 기준으로한 유사도가 가장 작아집니다. 벡터들의 원소값을 이용해서 사이각의 크기를 직접적으로 계산하는 것이 쉽지 않기 때문에 일반적으로 방향을 기준으로 한 유사도는 cosθ를 이용해서 계산됩니다. 사이각(θ)의 크기에 따라서 cosθ이 어떻게 달라지는지를 그림 7.17을 사용해서 살펴보겠습니다.

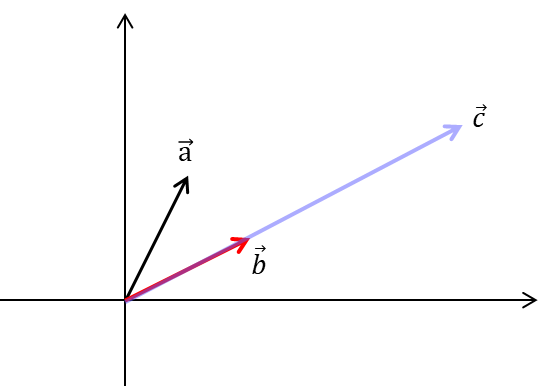


**그림 7.17 코사인 함수 (cosθ)**

사이각은 180˚가 넘어가면 다시 0˚과 180˚ 사이의 값으로 표현할 수 있기 때문에 그림 7.17에서 우리가 관심있는 구간은 θ의 값이 0˚과 180˚ 사이인 구간입니다. 그림 7.17에서 볼 수 있듯이 해당 구간에서 cosθ 값은 -1과 1사이의 값을 갖습니다. θ의 값이 0˚에 가까울수록 cosθ의 값이 증가하며 θ=0˚일 때 최댓값인 1의 값을 갖고, 180˚에 가까울수록 그 값이 작아져 θ=180˚ 일 때 최소인 -1의 값을 갖습니다. 즉, 사이각(θ)이 0˚에 가까울수록 cosθ 값이 증가하고 180˚에 가까울수록 cosθ 값이 감소하기 때문에 cosθ를 이용해서 방향을 기준으로한 두 벡터 사이의 유사도를 나타낼 수 있는 것입니다.

#### 코사인 유사도의 한계

**코사인 유사도는 두 벡터 사이의 사이각(즉, 방향의 유사도)만을 고려한 유사도**라는 것을 주의해야 합니다. 즉, 두 벡터 간의 거리는 크게 고려하지 않는 것입니다. 예를 들어서 길이가 서로 다른 세 개의 벡터가 그림 7.18과 같이 존재한다고 가정합니다.



**그림 7.18 서로 다른 세 개의 벡터**

이번에도 같은 질문을 던져 보겠습니다. 방향성을 기준으로 했을 때 벡터 는 벡터 와 중에서 어떤 벡터와 더 유사할까요? 방향성을 기준으로 했을 때는 와 의 유사도는 와 의 유사도는 동일합니다. 와 사이의 사이각과 와 사이의 사이각이 동일하기 때문입니다. 즉, cosθ 값이 같게 되는 것입니다. 와 의 길이가 다름에도 불구하고 말입니다.

그렇다면, 만약 유클리디안 거리를 사용해서 와 의 유사도, 와 의 유사도를 계산하면 어떻게 될까요? 이 경우에는 와 의 유사도가 더 크게 됩니다. 왜냐하면 와 사이의 거리가 더 짧기 때문입니다. 즉, 코사인 유사도는 벡터의 특성이 벡터의 길이 보다는 방향을 통해서 더 잘 표현되는 경우에 사용하는 것이 좋습니다. 그래서 많은 경우에 코사인 유사도는 벡터들 간의 길이를 동일하게 작업한 후에 수행하게 됩니다. 벡터들의 길이를 동일하게 하는 것을 정규화(normalization)라고 합니다. 보통은 길이를 1로 만들어 줍니다. 그리고 길이가 1인 벡터를 단위벡터 (unit vector)라고 합니다. 단위벡터는 벡터를 자기 자신의 길이로 나눠서 계산할 수 있습니다.

#### 단위벡터의 계산

라고 하는 경우에 의 단위 벡터는 와 같이 표현됩니다. 는 의 (원점으로 부터의) 길이를 의미합니다. 가 됩니다. 그러면 가 됩니다. 보는 것 처럼 의 길이, 즉 는 1이 됩니다. 그리고 의 방향은 원래 벡터인 와 같은 것을 알 수 있습니다.

#### 내적 공식을 이용한 cosθ 계산하기

그렇다면 벡터의 원소값을 이용해서 cosθ의 값을 어떻게 계산할 수 있을까요? 이를 위해서 내적의 개념을 활용합니다. 내적은 벡터 간의 곱하기라고 생각할 수 있습니다. 영어로는 dot product 혹은 inner product라고 합니다. 두 벡터 와 의 내적은 다음과 같이 정의 됩니다.

즉, 두 벡터 간의 내적은 같은 자리에 있는 원소들을 곱해서 더해준 것입니다. 내적의 결과는 하나의 숫자가 됩니다 (선형대수에서는 이러한 숫자를 스칼라라고 합니다). 구체적인 예를 들어보겠습니다. , 라는 두 벡터의 내적은 얼마가 될까요? 여러분이 직접 한번 계산해 보세요.

답은 얼마죠? 네 맞습니다, 6이 됩니다. 아래와 같이 계산됩니다.

파이썬에서는 numpy에서 제공되는 dot() 함수를 사용해서 벡터 간의 내적을 계산할 수 있습니다. 다음과 같이 사용합니다.

import numpy as np

a = np.array([1, 2])

b = np.array([2, 2])

np.dot(a, b)

6

그렇다면 우리는 내적을 이용해서 어떻게 cosθ 값을 구할 수 있을까요? 이를 위해서 다음의 내적 공식을 이용합니다.

는 의 길이 (즉, 원점으로부터의 유클리디안 거리)를 의미하고 는 와 의 사이각을 의미합니다. 두 벡터 와 에 대해서 , 가 됩니다. 위의 식의 양변을 로 나눠주면 아래와 같은 결과를 얻을 수 있습니다.

그리고 인 사실을 이용하면, 아래의 결과를 얻습니다.

이번에도 두 벡터 , 의 사이각에 대한 코사인 값을 계산해 보겠습니다.

이번에는 numpy를 이용해서 계산해 보겠습니다. 다음과 같이 코딩할 수 있습니다. np.dot(a,b)은 두 벡터 , 사이의 내적을, np.linalg.norm(a)은 벡터 의 길이를 계산합니다.

np.dot(a,b)/(np.linalg.norm(a)\*np.linalg.norm(b))

0.9486832980505138

#### 코사인 거리 (Cosine distance)

cosθ의 값을 사용한 코사인 거리 지표도 있습니다. cosθ 값은 그 값이 커질수록 벡터간의 유사도가 크다는 것을 의미합니다. 반대로 코사인 거리는 (거리이기 때문에) 값이 커질수록 유사도가 작아집니다.코사인 거리는 아래와 같이 정의 됩니다.

코사인 거리 (cosine distance) = 1 – cosθ

코사인 거리는 Scipy의 하위 모듈인 distance 모듈에서 제공되는 cosine() 함수를 사용해서 계산할 수 있습니다. 이 함수는 cosθ가 아니라 1- cosθ의 결과를 반환합니다. 주의하세요.

import scipy.spatial.distance as dst

dst.cosine(a,b)

위 코드는 0.05131670194948623의 값을 반환됩니다. 이는 1 – cosθ (즉, 1 - 0.948683)에 해당하는 값입니다.

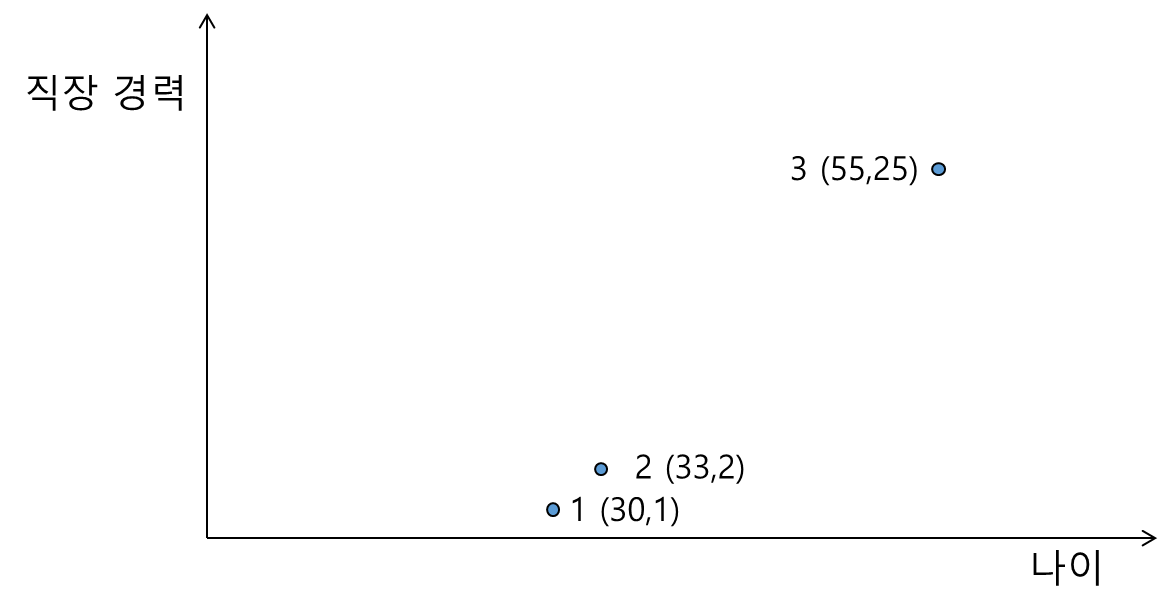
### 데이터 분석에서의 벡터

그렇다면 기계학습과 같은 실제 데이터 분석에서는 벡터가 어떻게 사용될까요? 설명을 위해 그림7.19와 같은 데이터가 있다라고 가정하겠습니다. 세 명의 사람들에 대한 나이 정보와 직장 경력 정보가 저장되어 있습니다. 우리는 보통 다음과 같은 데이터를 엑셀이나 csv 파일 형태로 저장하게 됩니다. 하지만, 기계학습을 이용하여 해당 데이터를 분석하기 위해서는 각 관측치를 벡터로 표현해야 합니다.

|  |  |  |
| --- | --- | --- |
| 관측치 | 나이 | 직장 경력 |
| 1 | 30 | 1 |
| 2 | 33 | 2 |
| 3 | 55 | 25 |

**그림 7.19 세 개의 관측치로 구성된 데이터**

관측치 1은 벡터 1 = (30, 1)로 표현할 수 있습니다. 즉, 첫 번째 특성 정보인 나이의 값이 첫 번째 원소가, 두 번째 특성 정보인 직장 경력의 값이 두 번째 원소가 됩니다. 해당 벡터는 원소를 두 개 갖고 있기 때문에 2차원 공간에 존재하는 하나의 점이 됩니다. 앞에서 설명한 것 처럼 공간 상의 하나의 점으로 표현되기 때문에 고유한 위치와 방향을 갖습니다. 그리고 그러한 위치와 방향은 벡터가 갖는 원소의 구체적인 값 (즉, (30, 1))에 의해 결정됩니다. 나머지 두 개의 관측치도 벡터 2 = (33, 2), 3 = (55, 25)로 표현됩니다. 이러한 벡터 정보를 기계학습 알고리즘이 분석에 사용하게 되는 것입니다. 세 개의 벡터를 공간 상에 표현해 보겠습니다 (그림 7.20 참고).



**그림 7.20 공간에서 세 관측치의 위치**

위의 그림을 보면 1번 관측치와 2번 관측치에 대한 벡터가 더 가까이 위치해 있는 것을 알 수 있습니다. 즉, 1번 관측치와 2번 관측치가 3번에 비해서 더 유사하다는 것을 의미합니다 (유클리디안 거리 기준으로 그러한 것입니다). 하지만 여러분이 유념해야 하는 것은 벡터로 표현되는 각 관측치의 특성은 각 관측치를 벡터로 표현할 때 사용된 특성 정보에 대한 것 뿐이라는 것입니다. 즉, 위의 예에서 벡터를 통해서 우리가 파악할 수 있는 관측치의 특성은 나이와 직장 경력에 대한 것입니다. 각 관측치가 갖는 그 외 다른 특성 (예, 성별, 정치 성향, 소득, 교육 정도 등)은 위 벡터를 통해서 표현되지 않고 그렇기 때문에 다른 특성을 기준으로한 관측치들 간의 유사도는 위 벡터들을 이용해서는 파악할 수 없습니다.

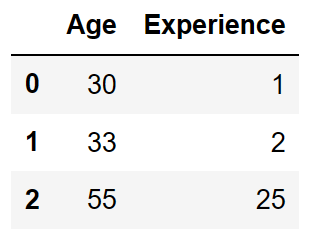
다시 정리하자면, 각 관측치를 벡터로 표현할 때 사용된 나이와 직장 경력 특성 기준으로는 1번 사람과 2번 사람의 유사도가 3번에 비해서 더 크다고 해석할 수 있습니다.

이번에는 csv 파일에 저장되어 있는 데이터를 파이썬으로 불러와서 앞에서 설명한 유사도 값을을 직접 계산해 보겠습니다. 위의 예제 데이터는 vector\_example.csv 파일에 저장되어 있습니다. csv 파일이나 엑셀 파일에 저장되어 있는 데이터를 파이썬으로 불러올 때 유용하게 사용할 수 있는 모듈이 pandas 입니다. pandas에서 제공되는 read\_csv() 함수를 이용해서 vector\_example.csv 파일의 내용을 다음과 같이 읽어옵니다.

import pandas as pd

data = pd.read\_csv('income\_toy.csv')

data 변수에는 다음과 같은 내용이 저장되어 있습니다.



**그림 7.21 data 변수에 저장된 데이터**

numpy 모듈에서 제공되는 함수들을 이용해서 벡터 관련 연산을 수행하기 위해서는 pandas를 이용해서 읽어온 데이터를 numpy의 어레이(array) 형태로 변환해야 합니다. 이를 위해서 아래와 같이 values 키워드를 사용합니다.

data\_np = data.values

data\_np에는 아래와 같이 어레이 형태의 데이터가 저장되어 있습니다.

array([[30, 1],

[33, 2],

[55, 25]], dtype=int64)

이러한 형태의 데이터를 행렬이라고 합니다. 행렬은 벡터를 쌓아 놓은 것이라고 생각할 수 있습니다. 즉, 첫 번째 행 ([30,1])이 첫 번째 관측치에 대한 벡터, 두 번째 행이 두 번째 관측치에 대한 벡터 (즉, [33, 20]), 세 번째 행이 세 번째 관측치에 대한 벡터가 되는 것입니다. 이러한 세 개의 벡터를 세로로 쌓아서 행이 3개, 열이 2개인 3x2 행렬을 만들었다고 생각할 수 있습니다.

각 관측치의 벡터를 추출하기 위해서 인덱싱을 사용합니다. 첫 번째 관측치에 대한 벡터는 data\_np[0], 두 번째는 data\_np[1] 등이 됩니다. 우리는 이러한 벡터 정보를 사용해서 벡터 간의 유사도를 계산할 수 있습니다. 여기서는 첫 번째 관측치의 벡터와 두 번째 관측치 벡터 간의 유클리디안 거리와 cosθ를 계산해 보겠습니다.

유클리디안 거리는 아래와 같이 구할 수 있습니다.

np.linalg.norm(data\_np[1]-data\_np[0])

3.1622776601683795

cosθ는 아래와 같이 구할 수 있습니다.

np.dot(data\_np[0],data\_np[1])/(np.linalg.norm(data\_np[0])\*np.linalg.norm(data\_np[1]))

0.999629802932285

마찬가지로 첫 번째 벡터와 세 번째 벡터의 유사도도 구할 수 있습니다 (직접 구해서 예제 코드에 있는 답과 맞춰 보세요). 유클리디안 거리, 코사인 유사도 모두 첫 번째와 두 번째 관측치가 더 유사하다는 것을 보여주고 있습니다.

## 확률의 이해

확률에 대해서 이해하기 위해서는 먼저 시행과 사건이라는 개념에 대해서 이해하는 것이 필요합니다. 왜냐하면 우리가 일반적으로 사용하는 '확률'이라고 하는 표현은 어떠한 사건이 발생할 확률을 의미하기 때문입니다.

### 시행과 사건

#### 시행 (Experiment 또는 Trial)

시행은 동일한 조건에서 반복해서 수행할 수 있고, 그 결과를 사전에 알 수 없는 행동을 의미합니다.[[17]](#footnote-18) 주사위를 던지는 행동, 동전을 던지는 행동들이 이에 해당 합니다. 시행의 결과값이 여러개 (두 개 이상)이고, 시행을 수행하기 전에 시행의 결과가 무엇인지 모르는 경우 (즉, 결과에 대한 불확실성이 존재하는 경우), 우리는 그러한 시행을 무작위 시행 (random experiment)라고 합니다. 확률에서 의미하는 시행은 일반적으로 무작위 시행을 의미합니다. 여러분들이 쉽게 생각할 수 있는 시행의 예로는 주사위를 한 번 던지는 시행, 두 개의 동전을 동시에 던지는 시행 등이 있습니다.

확률에 대해서 잘 이해하기 위해서는 집합에 대해서 아는 것이 필요한에, 그 이유는 많은 경우 확률에서의 중요한 개념들 (예, 시행 또는 사건 등)이 집합을 사용해서 표현되기 때문에 그렇습니다. 시행은 표본 공간이라는 집합으로 표현됩니다.

#### 표본 공간 (Sample space)

표본 공간은 어떠한 시행에서 일어날 수 있는 모든 발생가능한 결과의 집합을 의미합니다.[[18]](#footnote-19) 예를 들어, 주사위를 한 번 던지는 시행의 표본 공간은 {1, 2, 3, 4, 5, 6}이 되고, 동전을 한번 던지는 시행의 표본 공간은 {H, T}이 됩니다 (여기에서 H는 앞면 (head)를 T는 뒷면 (tail)를 의미합니다). 그렇다면, 두 개의 동전을 동시에 던지는 시행의 표본 공간은 어떻게 될까요? 이는 {(H,H), (H,T), (T,H), (T,T)} 가 됩니다.

표본 공간이 갖는 각 원소(element)는 시행의 특정한 결과를 의미하며, 하나의 원소를 표본점(sample point)이라고 표현합니다

#### 사건 (Event)

이번에는 사건에 대해서 알아보도록 하겠습니다. 사건은 시행의 결과로 나올 수 있는 특정한 값 또는 값들의 집합으로 정의 됩니다. 예를 들어 보겠습니다. 한 개의 주사위를 던지는 시행에 있어서는 다음과 같은 여러 가지의 사건이 존재할 수 있습니다 (아래 열거된 것 이외에도 더 있습니다).

* 1의 눈이 나오는 사건
* 2의 눈이 나오는 사건
* …
* 6의 눈이 나오는 사건

위의 사건들은 모두 시행의 결과로 나오는 값이 하나인 사건들입니다. 이번에는 시행의 결과로 나올 수 있는 값이 두 개 이상인 경우를 살펴 보겠습니다.

* 짝수의 눈이 나오는 사건
* 홀수의 눈이 나오는 사건

위의 사건들은 한 개의 주사위를 던지는 시행의 결과로 나올 수 있는 값의 수가 두 개 이상인 경우입니다. 짝수의 눈이 나오는 사건에 해당하는 결과값은 2, 4, 6으로 집합으로 표현하면 {2, 4, 6}이 됩니다. 홀수의 눈이 나오는 사건의 경우는 {1, 3, 5}와 같이 표현할 수 있습니다. 그리고 1의 눈이 나오는 사건은 {1}로, 2의 눈이 나오는 사건은 {2}와 같이 역시나 마찬가지로 집합으로 표현될 수 있습니다. 이와 같이 어떠한 시행에 대한 사건은 집합으로 표현될 수 있습니다.

그렇다면, '한 개의 주사위를 던지는 시행'에 대한 사건을 나타내는 집합들은 해당 시행의 표본 공간과 어떠한 관계를 갖을까요? 사건의 집합들 (예, )은 모두 이라고 하는 해당 시행에 대한 표본 공간의 부분 집합인 것을 알 수가 있습니다. 어떠한 시행의 특정한 사건을 나타내는 집합을 E라고 표현하고, 해당 시행에 대한 표본 공간을 S라고 표현한다면, E와 S 간에는 아래와 같은 관계가 성립하게 됩니다.

시행에 대한 사건은 수학적으로 다음과 같이 정의됩니다.

사건: 시행에 대한 표본 공간의 부분 집합

그리고 어떠한 시행의 결과값(표본 공간이 갖는 하나의 원소가 됩니다)이 사건 E 집합에 속하는 경우, 즉, 원소인 경우, 우리는 **사건 E가 발생했다**라고 표현합니다. 예를 들어, 한 개의 주사위를 던지는 시행에 대해 짝수의 눈이 나오는 사건을 E라고 가정합니다. E 집합은 다음과 같습니다.

그러한 경우에 주사위를 던져서 2의 눈이 나온다면, 2는 집합 E의 원소이기 때문에 “사건 E가 발생했다”라고 표현할 수 있는 것입니다.

**사건에서의 합집합과 교집합**

집합 A와 B가 사건인 경우, 해당 사건들의 합집합인 와 교집합인 도 사건이 됩니다 (즉, 합집합과 교집합 모두 표본공간의 부분집합이 됩니다). 예를 들어 주사위를 한 번 던지는 시행에 대해 다음과 같은 두 사건이 있다고 가정합니다.

사건 A: 짝수의 눈이 나오는 사건, 즉

사건 B: 5이상의 눈이 나오는 사건, 즉,

위 두 사건에 대한 합집합 ()은 이 되고, 교집합 ()은 이 됩니다. 과 모두 시행의 표본 공간인 의 부분 집합인 것을 알 수 있습니다. 따라서 두 사건 A, B의 합집합과 교집합도 또 다른 사건이 됩니다.

##참고 시작##

참고

두 사건 A, B의 교집합의 확률 즉, 은 두 사건이 동시에 발생할 확률을 의미하며, 또는 로 표현되기도 합니다. 반면에 두 사건 A, B의 합집합의 확률 즉, 은 사건 A 또는 사건 B가 발생할 확률을 의미하며 로 표현되기도 합니다.

## 참고 끝##

### 확률의 계산

우리가 일반적으로 얘기하는 확률은 어떠한 사건이 발생할 확률을 의미합니다. 어떠한 사건의 확률 (probability of an event)은 해당 사건이 일어날 수 있는 가능성을 수로 표현한 것입니다. 사건 A가 발생할 확률은 P(A)라고 표현합니다.

사건이 발생할 확률은 아래와 같이 정의 됩니다.

(특정 사건이 발생할) 확률 = 해당 사건에 대한 결과의 수 / 해당 시행 관련 모든 가능한 결과의 수

이를 집합으로 표현하면 “사건 A 집합의 원소의 수 / 표본 집합 S의 원소의 수”가 됩니다. 즉, 아래와 같이 표현됩니다.

여기서 는 사건 A 집합의 원소의 수를, 는 표본 공간 S가 갖는 원소의 수를 의미합니다. 구체적인 예를 몇 개 들어보겠습니다.

예1) 하나의 주사위를 던져서 짝수가 나올 확률

첫 번째 예로, 하나의 주사위를 던저셔 짝수가 나올 확률을 계산해 보겠습니다. 이러한 확률을 계산하기 위해서 먼저 해야하는 것은 시행과 사건을 명확하게 표현하는 것입니다. 이 예에서의 시행은 “하나의 주사위를 던지는 시행”이고, 사건은 “짝수가 나오는 사건”입니다. 이를 집합으로 표현하면 아래와 같습니다.

표본 공간 =

짝수가 나오는 사건에 대한 집합 =

따라서 표본 공간의 원소의 수 (즉, 시행 관련 모든 가능한 결과의 수)는 6이 되고, 사건에 대한 집합의 원소의 수 (즉, 사건에 대한 결과의 수)는 3이 되기 때문에 우리가 구하고자 하는 확률의 값은 가 됩니다.

예 2) 두 개의 동전을 동시에 던져서 둘 다 앞 면이 나올 확률

이번에는 두 개의 동전을 동시에 던져서 두 동전 모두 앞면이 나오는 확률을 구해보도록 하겠습니다. 이 예에서의 시행은 “두 개의 동전을 동시에 던지는 시행”이고 사건은 “ 두 동전 모두 앞면이 나오는 사건”입니다. 해당 시행에 대한 표본 공간 (S라고 표현하겠습니다)과 사건 집합 (E라고 표현하겠습니다)은 아래와 같습니다.

따라서 우리가 구하고자 하는 는 아래와 같이 계산됩니다.

### 확률에 있어 기본이 되는 공리 (axioms, 명제라고도 합니다)

확률의 기본적인 공리는 아래와 같습니다.

1) 공리 1: 어떠한 사건 A에 대해서

2) 공리 2: 표본공간 S의 확률 = 1, 즉,

3) 공리 3: 가 서로 상호배타적 사건(disjoint events)인 경우,

첫 번째 공리는 어떠한 사건의 확률은 음수가 될 수 없다는 것을 의미합니다. 어떠한 사건의 확률, 즉, P(A)가 취할 수 있는 가장 작은 값은 0이 되고, P(A)=0이라는 것은 사건 A가 발생하지 않았다는 것을 의미합니다.

두 번째 공리는 표본공간의 확률이 1이라는 것을 의미하는데, 이는 표본공간은 어떠한 시행으로 발생할 수 있는 모든 결과값을 원소로 갖는 집합이기 때문에 당연할 수 밖에 없습니다. P(S)=1 이라는 것은 어떠한 시행의 결과가 반드시 표본 공간 (S)의 원소가 된다는 것을 의미합니다.

세 번째 공리는 두 사건이 서로 상호배타적인 경우, 즉, 두 사건의 교집합이 공집합인 경우, 두 사건의 합집합의 확률은 각 사건의 확률을 더한 것과 같다는 것을 의미합니다. 예를 들어, 한 개의 주사위를 던지는 시행에 대해서 1의 눈이 나오는 사건 A와 2의 눈이 나오는 사건 B의 경우, 표본 공간과 각 사건의 집합은 아래와 같이 표현됩니다.

따라서 , 즉, 사건 A와 B는 서로 상호배타적 (disjoint) 입니다. 이러한 경우, 두 사건 A, B에 대해서 공리 3이 아래와 같이 성립되는 것으 알 수 있습니다.

공리 3을 아래와 같이 표현할 수 있습니다.

If

then

만약 어떠한 두 사건이 서로 상호배타적이 아니라면, 두 사건의 합집합에 대한 확률은 아래와 같이 계산됩니다.

앞의 3개의 공리를 사용해서 다음 식들을 증명해 보도록 하겠습니다.

1) 어떠한 사건 A에 대해서

2) 공집합의 확률은 0, 즉,

3) 어떠한 사건 A에 대해서

4)

5)

6) 만약 라면,

### 조건부 확률 (conditional probability)

두 개의 사건 A, B에 대해서, 사건 B가 발생했다는 조건 하에서 사건 A가 발생할 확률이 얼마인지를 계산해야 하는 상황이 있습니다. 이러한 확률을 조건부 확률이라고 하며, 라고 표현합니다. 만약에 A 사건이 B 사건의 발생 여부에 영향을 받는 사건이라고 한다면, 와 의 값은 다르게 됩니다.

예를 하나 들어보겠습니다. 한 개의 주사위를 던지는 시행에 대해서 홀수의 눈이 나오는 사건을 A라고 하고, 3 이하의 눈이 나오는 사건을 B라고 하겠습니다. 그렇다면 S={1,2,3,4,5,6}, A={1,3,5}, B={1,2,3}이 됩니다.

이러한 경우에 가 됩니다. 그렇다면, B가 발생했다라는 조건하에 A가 발생할 확률, 즉 는 얼마가 될까요? B가 발생했다라는 조건하에 A가 발생하는 경우, A 사건의 결과는 반드시 의 원소가 되어야 합니다. 그리고 사건 B가 발생했다는 조건이기 때문에, 는 다음과 같이 표현될 수 있습니다.

위 사건 A, B에 대해서 이기 때문에 이 되어 우리가 구하고자 하는 조건부 확률인 이 됩니다.

보다 일반적으로 는 아래와 같이 표현됩니다.

따라서 표본공간 S에 대한 두 개의 사건 A, B에 대해서, 사건 B가 발생했다는 조건하에서 사건 A가 발생할 확률 (즉, 조건부확률)은 다음과 같이 정의됩니다.

앞의 예, 즉, S={1,2,3,4,5,6}, A={1,3,5}, B={1,2,3}인 경우에 대해서 위의 식을 이용해서 를 다시 한 번 계산해 보겠습니다. 먼저 분자는 이 되고, 분모는 이 됩니다. 따라서 가 되는 것을 알 수 있습니다.

위에서 볼 수 있는 것 처럼 조건부 확률도 하나의 확률입니다. 따라서 우리가 앞에서 다룬 세 개의 공리도 조건부 확률에 적용됩니다.

조건부 확률에 대한 공리

1) 공리 1: 사건 A에 대해서

2) 공리 2: 사건 B 조건하에서의 B의 확률, 즉, P(B|B) = 1

3) 공리 3: 가 서로 상호배타적 사건(disjoint events)인 경우,

## 참고 시작 ##

참고

① 사건 A와 B가 서로 상호배타적일 때 (즉, 일 때), 는 얼마일까요? 우리는 앞의 내용을 통해서 라는 것을 알고 있습니다. 그런데, 분자에서 이 되기 때문에

인 것을 쉽게 알 수 있습니다. A와 B가 상호배타적이기 때문에 두 사건은 동시에 발생할 수 없습니다. 따라서, B가 발생했다라는 조건하에서는 A는 발생할 수 없기 때문에 이 됩니다.

② 그렇다면 인 경우에는 의 값은 얼마가 될까요?

사건 B가 A의 부분집합이라는 것은 사건 B가 발생하는 경우에는 반드시 사건 A가 발생한다라는 것을 의미합니다. 따라서 이 되어야 합니다. 정말 그런지 조건부확률 공식을 통해서 확인해 보겠습니다. 인 경우, 가 되기 때문에, 아래와 같은 식을 도출할 수 있습니다.

## 참고 끝 ##

조건부 확률의 공식을 이용해서 아래 문제를 풀어보도록 하겠습니다.

문제: 두 개의 주사위를 동시에 던지는 경우에 대해, 첫 번째 주사위의 눈을 N1라고 하고, 두 번째 주사위의 눈을 N2라고 하겠습니다. N1+N2 = 5라는 조건하에서 N1=2 또는 N2=2일 확률은 얼마일까요?

예시 답안

두 개의 주사위를 동시에 던지는 시행에 대한 표본 공간은 아래와 같고, 원소의 수는 36이 됩니다.

S = {(1,1), (1,2), …, (6,1), (6,2), (6,3), (6,4), (6,5), (6,6)}

N1=2 또는 N2=2을 사건 A라고 하고, N1+N2 = 5인 경우를 사건 B라고 하겠습니다. 각 사건은 다음과 같은 집합으로 표현됩니다.

A = {(2,1), (2,2), (2,3), (2,4), (2,5), (2,6), (1,2), (3,2), (4,2), (5,2), (6,2)}

B = {(1,4), (2,3), (3,2), (4,1)}

그리고 두 사건의 교집합은 아래와 같습니다.

={(2,3), (3,2)}

위의 문제에서 우리가 구해야하는 확률은 사건 B가 발생했다는 조건하에서 사건 A가 발생할 확률, 즉 가 됩니다. 조건부 확률 공식을 이용하면 아래와 같이 풀 수 있습니다.

### 조건부 확률에 대한 연쇄법칙 (Chain rule)

두 사건 A, B에 대해서, 조건부 확률 공식을 다시 한번 써 보도록 하겠습니다. B가 발생했다라는 조건하에서의 A의 확률은 아래와 같습니다.

위 식의 양변에 를 곱하면 를 구할 수 있습니다.

반대로 사건 A가 발생했다라는 조건하에서의 B의 확률은 아래와 같이 됩니다.

이를 통해서 를 도출할 수 있습니다.

즉, 가 되는 것입니다. 이러한 식은 조건부 확률과 각 사건이 발생할 확률을 알고 있는 경우, 교집합의 확률을 구하고자 할 때, 즉 사건 A, B가 동시에 발생할 확률을 구하고자 할 때 유용하게 사용할 수 있습니다.

이번에는 세 개의 사건 A, B, C에 대해서 세 개의 사건이 동시에 발생할 확률 즉, 아래 확률에 대해서 알아보겠습니다.

이는 로 표현될 수 있습니다. 여기서 를 하나로 사건으로 간주하여 사건 D라고 표현해 보겠습니다. 그러면, 다음과 같은 식을 도출할 수 있습니다.

여기에서 됩니다. 의 양변에 A를 조건으로 걸면, 가 됩니다. 따라서 우리는 아래와 같은 식을 얻을 수 있습니다.

여러 사건의 교집합에 대한 확률을 위와 같이 여러 개의 조건부 확률을 가지고 표현하는 것을 조건부 확률에 대한 연쇄 법칙이라고 합니다. 이를 n개의 사건에 대해서 일반화를 하면 아래와 같은 공식을 갖습니다.

위의 공식을 이용해서 예제 문제를 풀어보도록 하겠습니다.

문제: 공장에서 어떠한 제품 100개를 생산했는데, 그 중에 결함이 있는 제품이 5개라고 가정합니다. 100개의 제품 중에서 3개를 선택했을 때, 셋 모두 결함이 없는 제품일 확률은 얼마일까요?

예시 답안

첫 번째 제품이 결함이 없는 제품인 사건을 , 두 번째 제품이 결함이 없는 제품인 사건을 , 세 번째 제품이 결함이 없는 제품인 사건을 라고 하는 경우, 우리가 알고자 하는 값은 입니다. 이를 위에서 배운 연쇄법칙을 통해서 계산해 보겠습니다. 연쇄법칙을 사용하면

으로 표현됩니다.

는 100개의 제품 중에서 결함이 없는 95개의 제품을 뽑을 확률이므로, 이 됩니다. 는 사건 이 발생했다라는 조건하에서의 의 확률이므로 가 됩니다.[[19]](#footnote-20) 마찬가지로, 는 사건 과 가 발생했다라는 조건하에서의 의 확률이므로 이 됩니다. 따라서, 아래와 같은 결과를 얻게 됩니다.

### 독립 사건 (Independent events)

이번에는 서로 독립인 두 사건의 관계에 대해서 알아보도록 하겠습니다. 두 사건이 서로 독립이라는 것은 한 사건이 발생할 확률이 다른 사건의 발생 여부에 영향을 받지 않는다는 것을 의미합니다. 두 사건 A와 B가 서로 독립인 경우에, 사건 B가 발생할 확률은 사건 A에 의해서 영향을 받지 않는다는 것을 의미합니다. 즉, 는 사건 A의 발생 여부와 상관없이 동일한 값을 갖게 됩니다. 예를 들어, 두 개의 동전을 동시에 던지는 시행에 대해서 첫 번째 동전의 앞면이 나오는 사건을 A, 두 번째 동전의 앞면이 나오는 사건을 B라고 하겠습니다. 이러한 경우, 사건 A와 B는 서로 독립입니다. 즉, 하나의 사건이 발생할 확률이 다른 사건의 발생 여부에 영향을 받지 않습니다 (예, 사건 B의 확률은 로, 이는 첫 번째 동전의 앞면이 나왔는지와는 상관이 없습니다).

사건 B의 확률이 사건 A에 영향을 받지 않는다는 것을 수학적으로 표현하면 다음과 같이 됩니다.

이를 앞에서 배운 조건부 확률의 공식과 연결해 보겠습니다. 앞에서 다룬 조건부 확률의 공식은아래와 같습니다.

두 사건 A, B가 서로 독립인 경우에는 가 되기 때문에 아래와 같은 결과를 얻습니다.

위의 식 양변에 를 곱하면 아래의 식이 도출됩니다.

이는 사건 A와 B가 서로 독립인 경우에, “사건 A와 B가 동시에 발생할 확률은 각 사건의 발생 확률의 곱과 같다” 라는 것을 의미합니다.

사건 A와 B가 서로 독립인 경우에는 다음의 내용들도 만족합니다.

- A와 Bc가 서로 독립이다.[[20]](#footnote-21)

- Ac와 B가 서로 독립이다.

- Ac와 Bc가 서로 독립이다.

주의!

상호 배타적 (disjoint or exclusive) vs. 상호 독립적 (independent)

우리는 앞에서 두 사건이 상호 배타적인 것과 상호 독립적인 것에 대해서 배웠습니다. 둘의 차이를 명확하게 기억하는 것이 필요합니다. 상호 배타적이라는 것은 두 사건의 교집합이 공집합이다라는 것을 의미하고, 이는 두 사건이 동시에 발생하지 않는다는 것을 의미합니다. 반면에 두 사건이 서로 독립이라는 것은 한 사건이 발생하는 확률이 다른 사건에 의해서 영향을 받지 않는다는 것을 의미합니다.

사건 A, B에 대해서

그리고 인 경우, 두 사건이 서로 배타적인 경우, 두 사건은 서로 독립이 될 수 없습니다. 왜냐하면, 두 사건이 서로 배타적인 경우 이 되는 반면, 두 사건이 서로 독립이 되는 경우에는 이 되기 때문입니다.

### 조건부 독립 (conditional independence)

앞에서 다룬 독립 확률에 대한 대부분의 내용이 조건부 확률에 대해서도 적용이 됩니다. 두 사건 A, B가 독립인 경우 다음과 같은 관계가 성립되는 것을 다시 한 번 기억하길 바랍니다.

또는

두 사건 A, B가 사건 C가 발생했다는 조건하에서 서로 독립인 경우에는 아래와 같은 관계가 성립합니다.

두 사건 A, B에 대한 조건부 확률은 다음과 같이 정의가 됩니다.

이러한 조건부 확률의 특성은 또 다른 사건 C에 대해서 다음과 같이 확장됩니다.

위 내용은 아래와 같이 증명될 수 있습니다 (앞에서 언급한 것 처럼 는 두 사건 B, C가 동시에 발생할 확률을 의미하며 이는 와 동일한 표현입니다).

위의 식에서 분자인 는 다음과 같이 표현됩니다.

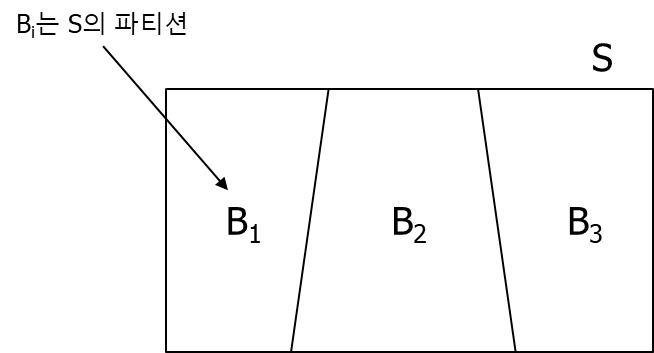
그리고 분모인 가 됩니다. 따라서 우리는 아래와 같은 식을 갖게 됩니다.

C 조건하에서 A와 B가 서로 독립인 경우에는 따라서 아래 관계가 성립하게 됩니다.

### 전체 확률의 법칙 (Law of total probability)

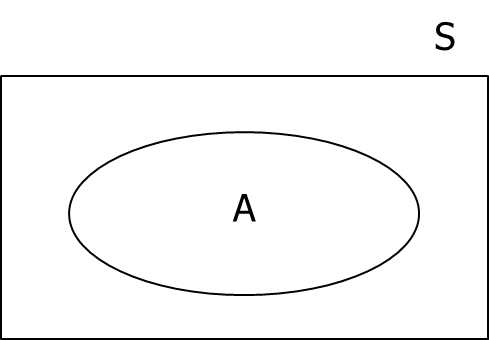
이번에는 전체 확률의 법칙에 대해서 살펴보겠습니다.

설명을 위해서 그림 10.1과 같이 표본 공간 S가 세 개의 서로 다른 부분 집합으로 구분된다고 가정합니다. 그림 10.1에서의 를 집합 S의 파티션(partition)이라고 합니다. 즉, 는 서로 상호배타적이어야 하고 (즉, 교집합이 없어야 하고), 전체의 합이 집합 S (즉, )가 되는 경우, 를 집합 S의 파티션이라고 합니다.



**그림 10.1 표본 공간의 파티션**

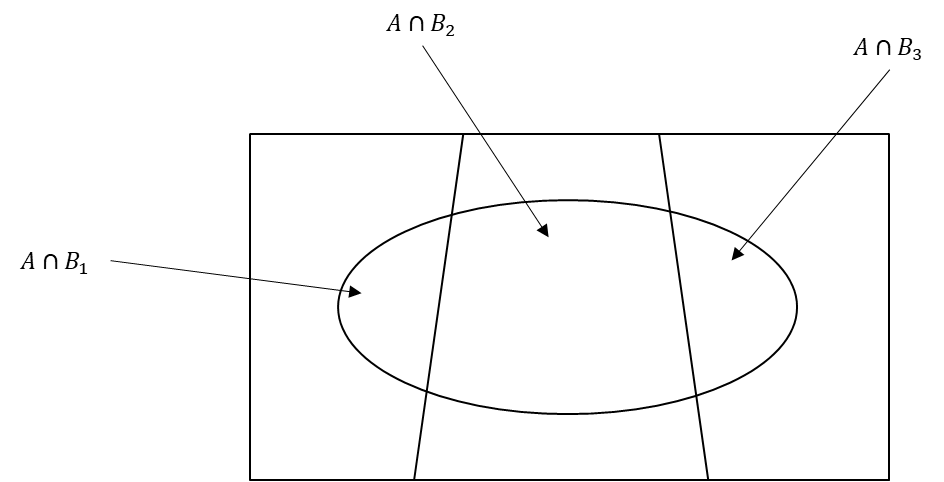
추가적으로 표본 공간 S에 대해서 그림 10.2와 같은 사건 A가 있다고 가정합니다 (사건 A는 S의 부분집합입니다).



**그림 10.2 사건 A**

이러한 경우 사건 A의 확률은 다음과 같이 표현될 수 있습니다.

이는 그림으로 표현하면 그림 10.3과 같습니다.



**그림 10.3 사건 A와 의 교집합**

그리고 조건부 확률 공식에 의해서 가 됩니다.

위의 관계를 일반화하면 아래와 같이 표현됩니다.

만약 이 표본 공간 S의 파티션이라면 사건 A에 대해서 다음 관계가 성립합니다.

이를 전체 확률의 법칙 (Law of total probability)라고 합니다.

### 베이즈 규칙 (Bayes’ rule)

베이지안 통계(Bayesian statistics) 혹은 베이지안 추론 (Bayesian inference)는 여러가지 통계분석 방법 혹은 기계학습 알고리즘에서 사용됩니다. 이러한 것의 바탕이 되는 것이 베이즈 규칙(Bayes’ rule)이 됩니다. 베이즈 규칙은 조건부확률을 기반으로 합니다.

조건부확률은 아래와 같습니다.

여기서 로 표현될 수 있습니다. 따라서 우리는 아래와 같은 식을 얻습니다.

이 식이 바로 베이즈 규칙이 되는 것입니다.

위 식에서 분모에 해당하는 는 전체확률 법칙에 따르면 아래와 같이 표현됩니다.

즉,

가 됩니다.

### 변수 (Random variables)

하나의 변수는 그 값이 무작위 시행 (random experiment)에 의해서 결정이 되는 것이라고 생각할 수 있습니다. 예를 한번 들어보겠습니다. 두 개의 동전을 동시에 던지는 시행에 대해서 우리는 다음과 같은 표본 공간을 갖습니다.

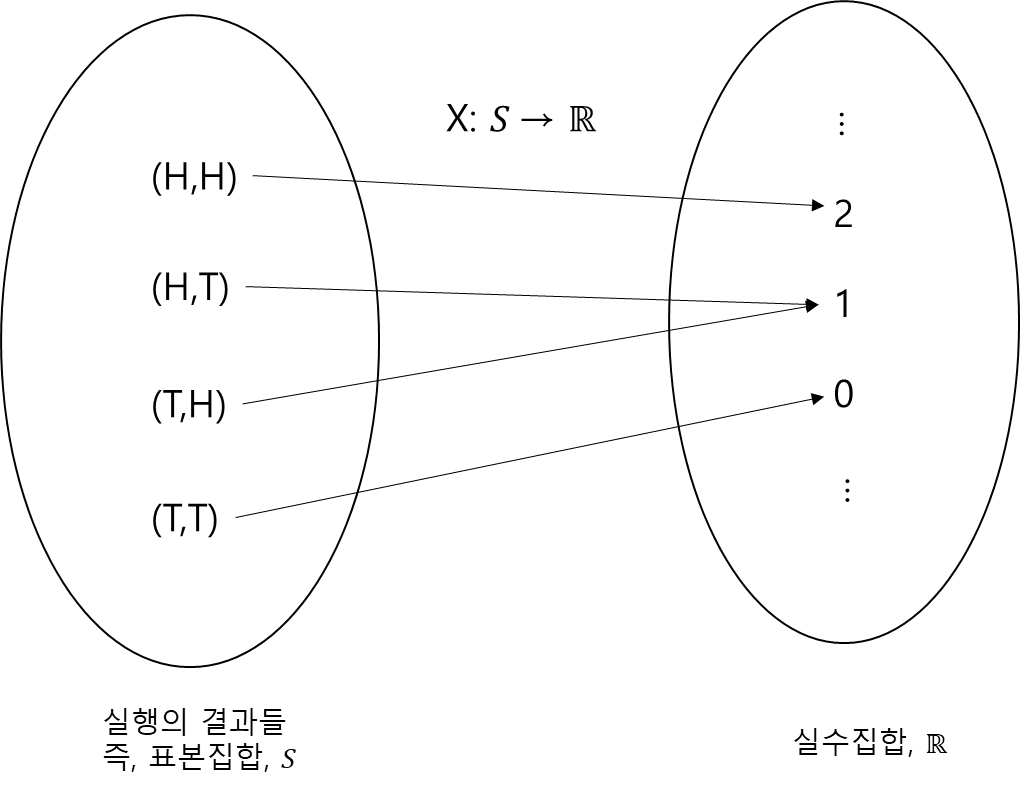
이러한 시행에 대해서 변수 X를 두 개의 동전을 동시에 던질 때 나오는 앞면의 수라고 정의하는 경우에, 변수 X가 취할 수 있는 값은 0, 1, 2가 됩니다. 그리고 이러한 값들은 시행의 결과에 의해 결정되게 됩니다. 예를 들어 시행의 결과가 인 경우에는 X의 값이 0이 되고, 그 결과가 또는인 경우에는 X의 값이 1, 인 경우에는 그 값이 2가 됩니다.

이러한 관점에서 변수는 하나의 숫자를 무작위 시행의 각 결과에 할당 (assign)하는 함수라고 간주될 수 있습니다. 예를 들어서, 위의 경우 변수 X는 라고 하는 시행의 결과에 대해서는 0을 와에 대해서는 1을, 에 대해서는 2라고 하는 숫자를 할당한다고 생각할 수 있습니다.

변수에서 다루는 숫자는 실수입니다. 따라서 일반적으로 변수 (random variable)는 “시행에 대한 표본 공간에서 실수로의 함수로 정의”됩니다. 즉, 아래와 같이 표현될 수 있습니다.

여기서 은 실수집합이 됩니다.

위 예에서의 변수 X를 그림으로 표현하면 그림 10.4와 같습니다.



**그림 10.4 변수**

변수는 간단하게는 여러 개의 값들 중에서 하나의 값을 취하는 어떤 것이라고 생각할 수 있습니다. 그런데, 그 취하는 값이 특정한 무작위 시행에 의해서 결정되는 것입니다.

변수가 취할 수 있는 각 값은 시행에 대한 특정한 사건과 대응된다고 생각할 수도 있습니다. 예를 들어, 앞에서 다룬 변수 X (즉, 동전을 두 개 던질 때 나오는 앞면의 수)가 취할 수 있는 값은 0, 1, 2 입니다. X=0인 경우는 (동전을 두 개 던질 때) 앞면의 수가 0인 사건, X=1은 앞면의 수가 1인 사건, X=2인 경우는 앞면의 수가 2인 사건과 대응된다는 것을 알 수 있습니다. 또 다른 예를 들어 보겠습니다. 이번에는 주사위를 한 개 던질 때 나오는 눈의 값을 변수 Y라고 하겠습니다. 이러한 경우 Y가 취할 수 있는 값들은 1, 2, 3, 4, 5, 6이 됩니다. Y=1인 경우는 (주사위를 한 개 던질 때) 나오는 눈이 1인 사건, … Y=6인 경우는 나오는 눈이 6인 사건이 되는 것입니다.

변수가 특정한 값을 취하는 확률을 계산해 보겠습니다. 변수가 특정한 값을 취하는 것은 하나의 사건이기 때문에 변수가 특정한 값을 취하는 확률은 앞에서 공부한 사건의 확률을 구하는 것과 동일한 방식으로 계산됩니다.

변수 X가 특정한 값 (이를 라고 표현하겠습니다)을 취할 확률 은 다음과 같이 계산이 됩니다 (이 하나의 사건이 됩니다).

에 해당하는 원소의 수 / 시행에 대한 표본공간의 원소의 수

또는

에 해당하는 결과의 수 / 시행으로 나올 수 있는 모든 결과의 수

예를 들어보겠습니다. 이번에도 앞에서 살펴본 변수 X (즉, 동전 두 개를 동시에 던졌을 때 나오는 앞면의 수)를 사용하도록 하겠습니다. 변수 X에 대한 시행은 “동전 두개를 동시에 던지는 것” 입니다. 그리고 이 시행에 대한 표본공간은 아래와 같이 됩니다.

변수 X가 취할 수 있는 값들은 {0, 1, 2}이 됩니다. 따라서 아래와 같이 표현될 수 있습니다.

이러한 경우에 변수 X가 특정한 값을 갖을 확률은 아래와 같이 표현됩니다.

X가 취할 수 있는 값이 0, 1, 2 이므로 우리는 을 구할 수 있습니다. 그리고, 0, 1, 2 이외의 다른 값들을 취할 수 없으므로, 즉, 0, 1, 2 값들 중에 하나를 반드시 취해야 하므로 이 되어야 합니다.

의 값을 구해보도록 하겠습니다. 이는 다음과 같이 계산됩니다.

인 사건에 해당하는 원소의 수 / 시행에 대한 표본공간의 원소의 수

인 사건은 집합 이기 때문에 분자의 값은 2가 됩니다. X에 대한 시행의 표본 공간은 이므로 분모의 값은 4가 됩니다. 따라서 가 됩니다.

비슷한 방식으로 가 되는 것을 알 수 있습니다. 따라서

이 됩니다.

**서로 독립인 변수들**

우리는 앞에서 서로 독립인 두 사건에 대해서 공부를 했습니다. 두 사건 A, B가 독립인 경우, 사건 A가 발생할 확률은 사건 B 발생 여부에 영향을 받지 않습니다. 이를 수학적으로 표현하게 되면

또는

가 됩니다.

비슷하게 서로 독립인 두 변수 X, Y에 대해서도 다음 관계가 성립합니다.

또는

이는 변수 X가 어떠한 값을 취할 확률은 변수 Y가 어떠한 값을 갖는지와 상관이 없다, 또는 영향을 받지 않는다는 것을 의미합니다 (X가 의 값을 취하는 사건을 A, Y가 의 값을 취하는 사건을 B라고 표현하면, 두 사건이 독립인 경우와 동일한 의미를 갖는 것을 알 수 있습니다).

예를 들어, 동전 한 개를 던지는 시행과 주사위 한 개를 던지는 시행에 대해서 서로 다른 두 개의 변수 X, Y를 다음과 같이 정의합니다.

변수 X: 동전을 던져서 나오는 앞면의 수

변수 Y: 주사위를 던져서 나오는 눈의 값

이러한 경우 X와 Y는 서로 독립인 관계가 됩니다. 왜냐하면 변수 X의 값이 어떠한 값을 갖는지가 변수 Y가 어떠한 값을 갖는지에 영향을 주지 않기 때문입니다 (그 반대도 성립합니다). 따라서 아래의 관계가 성립합니다.

그렇다면 동전을 던져서 나오는 앞면의 수가 0이고, 주사위를 던져서 나오는 눈이 6일 확률은 어떻게 계산할 수 있을까요? 아래와 같이 계산할 수 있습니다.

### 변수의 종류: 이산변수(discrete variable)와 연속변수(continuous variable)

변수는 이산변수와 연속변수로 구분됩니다. 이산변수는 변수가 취할 수 있는 값의 수를 셀 수 있는 경우 (즉, countable 한 경우)를 말합니다. 값의 수를 셀 수 있는 경우 보통은 그 값의 수가 유한(finite)합니다(예, 자동차의 바퀴수, 성별, 폐암 여부 등). 그래서 일반적으로 이산변수는 취할 수 있는 값의 수가 유한한 변수를 의미합니다. 하지만, 셀수 있지만, 취할 수 있는 값의 수가 무한한 경우도 있습니다 (영어로는 countably infinite이라고 표현합니다). 이러한 경우도 엄격하게 말하면 셀 수 있는 경우이기 때문에, 이산변수가 됩니다. 예를 들어, 1년에 콜센터에 걸려오는 전화의 수, 백화점을 방문하는 고객의 수, 1년에 발생하는 교통사고의 수, 하루에 새롭게 태어나는 개미의 수 등이 그러한 예입니다. 실질적으로는 그렇지 않겠지만, 이론적으로 이러한 변수들이 취할 수 있는 값의 수는 무한합니다.

반대로 연속변수는 취할 수 있는 값의 수가 무한하고(infinite) 셀수 없는(uncountable) 변수를 말합니다. 일반적으로 온도, 무게, 길이 등 (이론적으로) 특정한 구간에 존재하는 모든 실수값을 취할 수 있는 변수를 의미합니다. 즉, 특정 구간에 존재하는 값의 수 무한히 많은 경우 그러한 변수를 연속 변수라고 하겠습니다. 예를 들어, ‘길이’라는 연속 변수의 경우 10cm와 11cm 사이에 존재하는 실수의 값은 무한히 많습니다 (예, 10.1, 10.11, 10.111, 10.1111, …).

#### 이산변수의 확률

변수가 특정한 값을 취할 확률을 나타낼 때 일반적으로 함수를 사용합니다. 이산변수의 확률을 나타낼 때 사용되는 함수를 확률질량함수 (probability mass function, PMF)라고 하고, 연속변수의 확률을 나타낼 때 사용되는 함수를 확률밀도함수 (probability density function, PDF)라고 합니다.

확률질량함수 (probability mass function)

취할 수 있는 값들이 인 이산변수 X에 대해서, 확률질량함수는 아래와 같이 표현됩니다.

여기서 아래첨자 X는 가 변수 X에 대한 확률질량함수라는 것을 의미합니다. 이는 간단하게 아래와 같이 표현되기도 합니다.

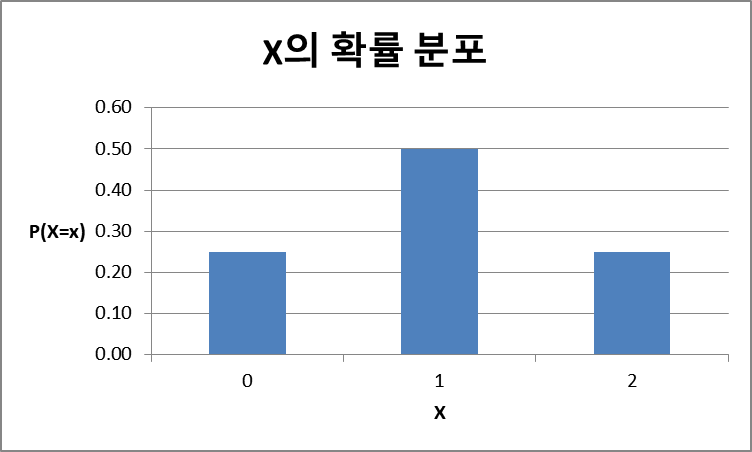
위에서 볼 수 있는 것 처럼, 이산변수 X의 확률질량함수의 값은 X가 특정한 값을 취할 확률이 됩니다.

확률질량함수의 구체적 예를 살펴보겠습니다. 이를 위해 앞에서 다뤘던 두 개의 동전을 던지는 시행에서 나오는 앞면의 수를 변수 X라고 하겠습니다. 변수 X가 취할 수 있는 0, 1, 2의 세 개로 그 수를 셀 수 있어 X는 이산변수가 됩니다. 변수 X의 확률질량함수는 아래와 같습니다.

X의 확률질량함수를 모든 숫자에 대해서 표현한다면, 아래와 같이 표현할 수 있습니다.

이산변수의 확률질량함수는 이산변수의 확률분포를 의미합니다. 하나의 변수가 취할 수 있는 각 값과 해당 변수가 각 값을 취할 확률을 대응시켜 놓은 것을 해당 변수의 확률분포라고 합니다

이를 그래프로 나타내면 아래와 같습니다.



**그림 10.5 이산변수 X의 확률분포 (PMF)**

이산변수의 확률질량함수의 또 다른 예를 살펴보겠습니다. 동전을 한 번 던졌을 때 앞면이 나올 확률이 인 동전이 하나 있다고 가정합니다 (즉, ). 이 동전을 앞면이 처음 나올 때 까지 동전을 반복해서 던지는 시행에 대해서 변수 Y를 다음과 같이 정의합니다.

변수 Y: 앞면이 처음 나올까지 동전을 던지는 횟수

변수 Y가 취할 수 있는 값들은 자연수가 됩니다. Y는 취할 수 있는 값의 수가 무한하지만 셀 수있는 (countably infinite) 변수이기 때문에 이산변수가 됩니다. 이산변수 Y의 확률질량함수를 찾아보도록 하겠습니다. 즉, Y에 대해서 를 찾아야 하는 것 입니다. 각 k의 값에 따라서 가 어떻게 되는지를 계산해 보겠습니다. 동전을 한 번 던질 때 앞면이 나올 확률이 p라는 것을 기억해야 합니다. 이는 뒷면이 나올 확률이 1-p라는 것을 의미합니다.

k=1 인 경우는 동전을 한 번 던졌는데, 바로 앞면이 나온 경우를 의미합니다. 따라서, Y의 값이 1이 되는 확률은 처음 동전을 던져서 앞면이 나올 확률과 동일합니다.

Y의 값이 2가 되는 상황을 생각해 보겠습니다. 이는 동전을 처음 던졌을 때는 뒷면이 나오고 두번째 던졌을 때 앞면이 나오는 상황을 의미합니다. 즉, P(Y=2)는 처음 던지기에서 뒷면이 나올 확률 곱하기 두 번째 던지기에서 앞면이 나올 확률이 됩니다. 즉,

Y=3의 경우는 처음 두 번의 던지기에서는 모두 뒷면이 나오고 마지막 세 번째 던지기에서 앞면이 나오는 상황입니다. 이는 다음과 같이 표현될 수 있습니다.

이런 과정을 일반화해 보면, Y=k의 경우는 처음 k-1번의 던지기에서는 모두 뒷면이 나오고 마지막 k 번째 던지기에서 앞면이 나오는 상황입니다. 이를 확률로 표현하면 아래와 같습니다.

따라서 Y의 확률질량함수는 다음과 같이 표현될 수 있습니다.

#### 이산 분포의 예: 베르누이 (Bernoulli) 분포

취할 수 있는 값이 0과 1인 변수를 베르누이 변수라고 하고,[[21]](#footnote-22) 해당 변수는 베르누이 분포를 갖는다고 표현합니다.

변수 X가 베르누이 분포를 따른다고 가정하고, 로 표현하는 경우, 베르누이 분포의 확률질량함수는 아래와 같이 정의됩니다.

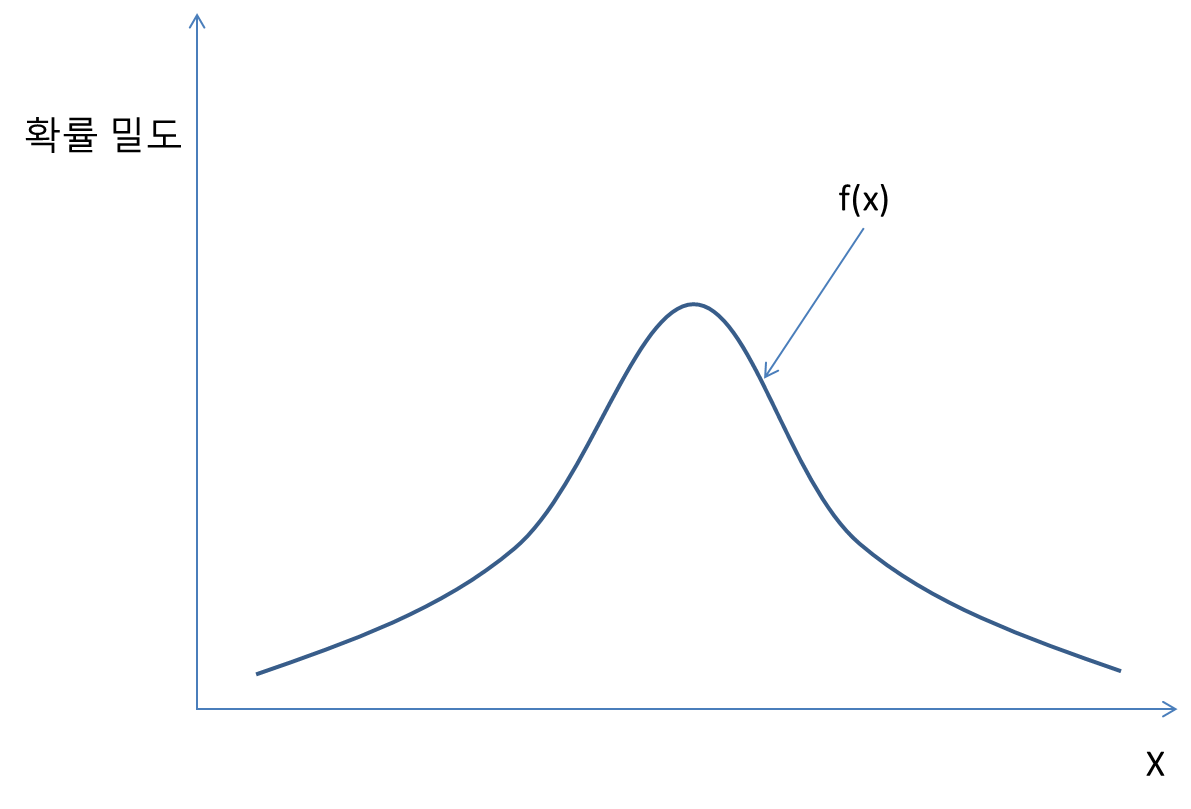
이는 변수 X가 취하는 값이 1인 경우의 확률이 이고, 0인 경우의 확률이 가 된다라는 것을 의미합니다. 0과 1이 아닌 다른 값까지 포함하면 아래와 같이 표현됩니다.

#### 연속변수의 확률

앞에서 말한 것 처럼, 연속변수는 취할 수 있는 값들이 셀수 없는 (uncountable) 변수를 의미합니다. 연속변수의 경우, 변수가 특정한 하나의 값을 갖을 확률은 0으로 정의됩니다. 연속변수 X에 대해서 이 되는 것입니다.[[22]](#footnote-23) 연속변수의 경우에는 변수가 특정한 값을 갖을 확률보다 변수가 특정 구간 사이의 값을 취할 확률 (혹은 변수가 취하는 값이 특정 구간 사이에 존재할 확률)이 중요합니다. 확률변수에 대해서 변수가 와 사이 ()의 값을 취할 확률은 아래와 같이 정의됩니다.

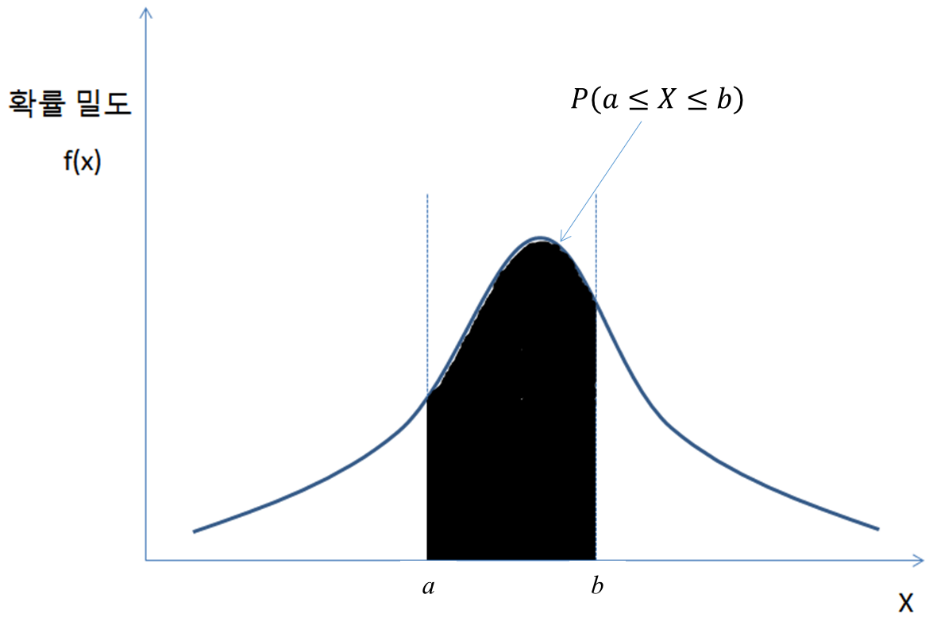
이때, 사용되는 을 확률밀도함수(probability density function, PDF)라고 합니다.[[23]](#footnote-24) 아래첨자 X는 해당 함수가 변수 X의 확률밀도함수라는 것을 의미합니다. 는 적분(integral)을 의미하며, 이는 의 아래부분 중에서 와 사이의 영역의 면적을 나타냅니다.

확률밀도함수는 그림 10.6과 같이 표현됩니다. 가로축이 연속변수 X가 취하는 값을 나타내고, 세로축이 확률밀도함수의 값을 나타냅니다. 변수 X가 특정한 값 (예, )를 취하는 경우의 확률밀도 값은 가 됩니다. 연속변수의 경우, 확률밀도함수가 변수의 확률 분포 형태를 결정하기 때문에, 확률밀도함수 자체가 연속 변수의 분포를 의미합니다.



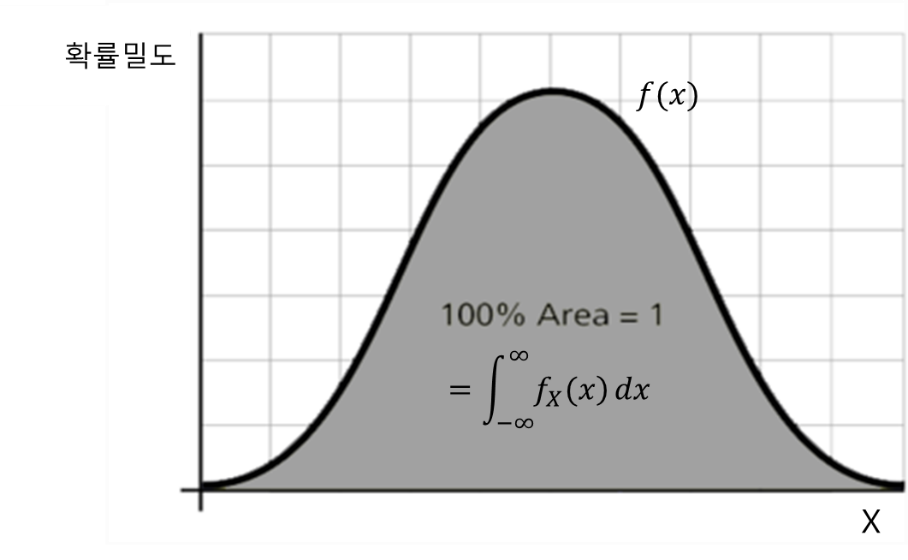
**그림 10.6 확률밀도함수의 예**

연속 변수가 특정 구간의 값을 취할 확률 (즉, 변수가 취하는 값이 특정 구간에 존재할 확률)은 확률밀도함수 아래의 면적으로 표현됩니다. 예를 들어, 를 그림으로 표현하면 그림 10.7과 같습니다.



**그림 10.7 의 의미**

확률밀도함수를 나타내는 곡선 아래 부분 전체 영역의 크기는 1이 됩니다. 이는 이산변수의 경우, 이산변수가 취할 수 있는 모든 값에 대한 확률의 합과 같습니다. 즉, 연속변수 X가 취할 수 있는 모든 값들에 대한 확률의 합은 1이라는 것을 의미합니다 (그림 10.8 참고). 이는 을 의미합니다.



**그림 10.8 확률밀도함수 아래 면적 전체의 크기는 1**

의 구체적인 값을 계산하기 위해서는 의 구체적인 형태를 알고 있어야 합니다. 하지만, 대부분의 경우에 우리는 특정한 변수가 갖는 확률밀도함수가 무엇인지 정확하게 알수가 없습니다. 그리고 변수마다의 고유한 확률밀도함수를 갖습니다. 확률 밀도 함수가 정확하게 어떠한 모양인지 알지 못한다면, 우리는 해당 변수의 값이 특정 구간에 위치할 확률을 구할 수가 없습니다.

### 확률밀도함수의 예: 정규분포의 확률밀도함수

#### 정규분포 (Normal distribution)

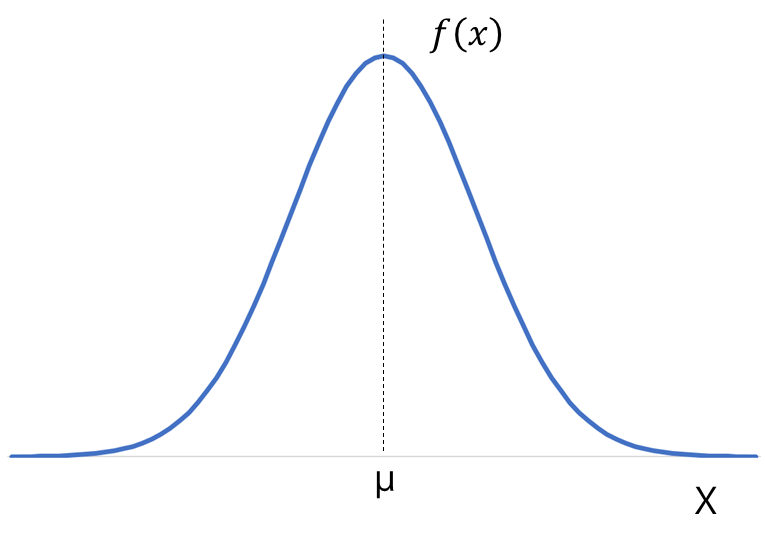
우리가 관심있어 하는 연속 변수가 특정 구간 사이의 값을 취할 확률을 계산하기 위해서 우리는 많은 경우에 하나의 연속 변수가 특정한 종류의 확률밀도함수를 갖는다고 가정을 합니다. 확률밀도함수들 중에서 가장 중요하고 가장 많이 사용되는 것이 정규분포 (Normal distribution)에 대한 확률밀도함수입니다. 정규 분포의 확률밀도함수는 다음과 같이 정의됩니다.

여기서 μ는 해당 분포의 평균을 의미하고 σ는 표준편차를 의미합니다.[[24]](#footnote-25) 확률 분포 (즉, 확률밀도함수)의 형태와 위치를 결정하는 확률밀도함수가 갖는 주요한 값들을 해당 분포의 파라미터라고 합니다. 정규분포 같은 경우에는 두 개의 파라미터가 존재하는데 μ와 σ입니다. 즉, μ와 σ가 구체적으로 취하는 값에 의해서 정규분포 (즉, 확률밀도함수)의 위치와 형태가 달라집니다.

연속변수 X가 평균이 μ이고 분산이 σ2인 정규분포를 따르는 경우 우리는 다음과 같이 표현합니다. N은 Normal distribution을 의미합니다.

변수 X는 평균이 μ이고 분산이 σ2인 정규분포를 갖는다라고 표현합니다.

정규 분포의 확률밀도함수는 그림 10.9와 같이 종 모양 (Bell shape)으로 평균을 중심으로 좌우가 대칭인 형태를 갖습니다.



**그림 10.9 정규분포의 형태**

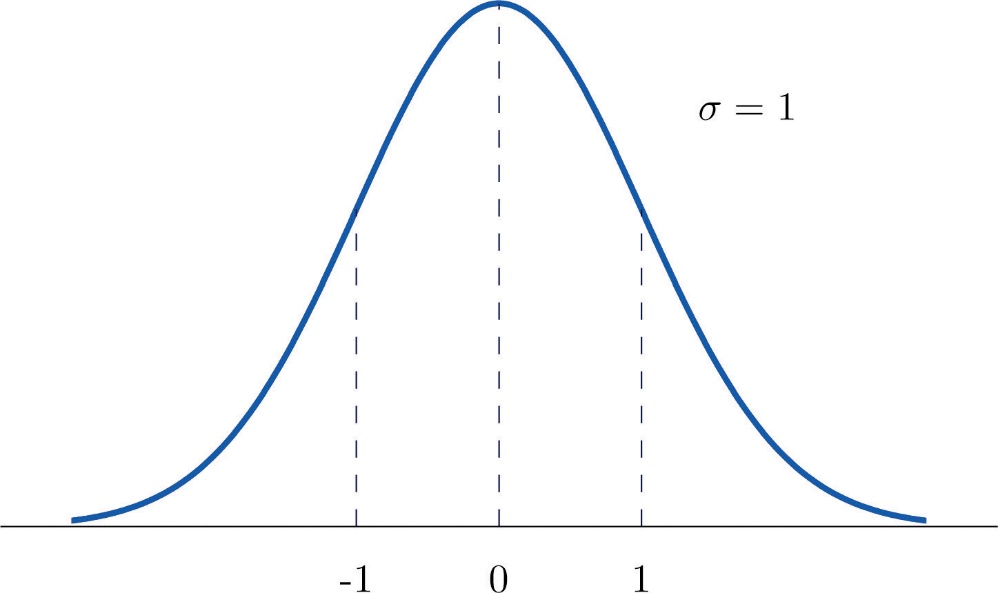
#### 표준정규분포 (Standard normal distribution)

위에서 설명한 것 처럼 우리는 특정 변수 (예, X)의 확률밀도함수를 사용해서 해당 변수의 값이 특정 구간 안에 포함될 확률을 구할 수가 있습니다. 하지만 이를 위해서는 해당 변수의 확률밀도함수가 정확히 어떠한 식으로 표현되는지를 알아야 합니다. 많은 경우 계산의 용이성을 위해서 하나의 변수는 정규분포를 따른다고 가정을 합니다. 왜냐하면 정규분포의 확률밀도함수는 다음과 같이 수학적 식으로 표현이 되기 때문에 이러한 공식을 이용해서 특정 영역의 크기를 계산할 수 있고, 정규분포는 평균을 중심으로 좌우가 대칭이라는 특성을 갖기 때문입니다.

하지만 이러한 계산도 사람이 직접 손으로 하기에는 많은 어려움이 있습니다. 정규분포를 갖는 연속변수의 값이 특정 구간에 포함될 확률을 보다 손쉽게 계산할 수 있는 방법은 평균이 μ이고 분산이 σ2 인 정규분포를 (즉,)를 평균이 0이고 분산이 1인 정규분포로 바꿔서 계산하는 것입니다. 평균이 0이고 분산이 1인 정규분포를 우리는 표준정규분포라고 부릅니다. 그리고 이라고 표현합니다. 즉, 의 분포를 갖는 X 변수의 분포를 표준화 시켜서 의 분포를 갖도록 만들고 를 이용해서 를 구하게 됩니다. 분포를 로 변환하기 위해서는 X 변수를 표준화해야 합니다. 이는 아래와 같은 공식을 통해서 할 수 있습니다.

이를 표준화 값 또는 Z score라고 부릅니다. 그래서 보통 아래와 같이 표현합니다.

와 같이 표현됩니다. 표준정규분포는 그림 10.10과 같습니다.

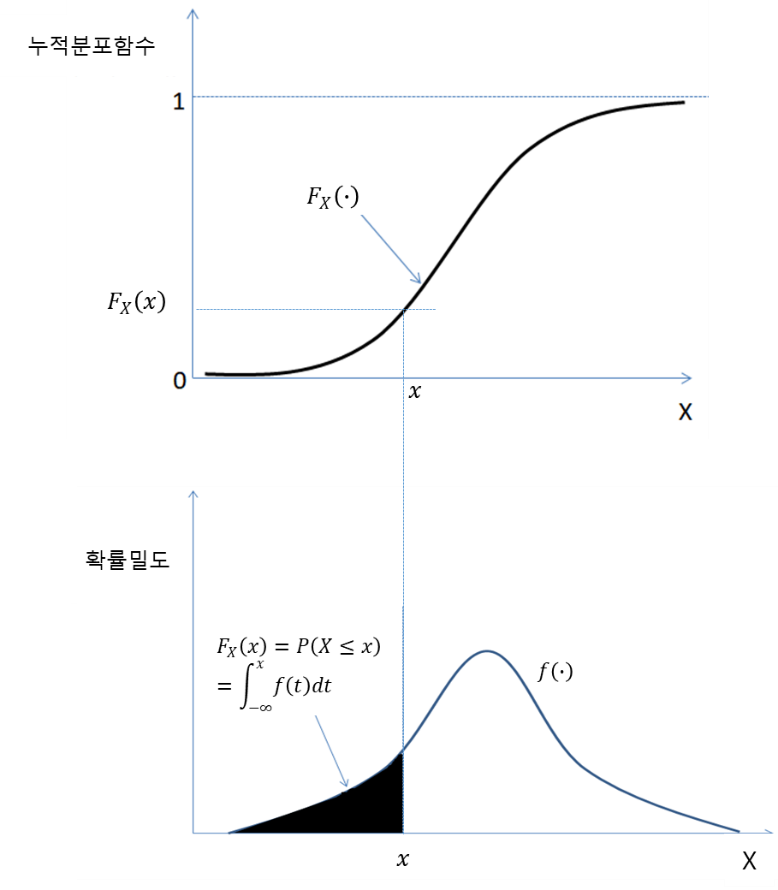


**그림 10.10 표준정규분포**

### 누적분포함수 (cumulative distribution function, CDF)

변수의 누적분포함수는 변수가 특정한 값 이하의 값을 취할 확률을 나타내는 함수로 0과 1 사이의 값을 갖습니다. 여기서는 연속변수에 대한 누적분포함수를 먼저 설명하겠습니다. 연속변수 X의 누적분포함수는 아래와 같이 정의됩니다.

누적분포함수는 와 같이 대문자를 이용해서 표현합니다. 이는 위의 식에서 알 수 있듯이 변수 X가 특정한 값 () 보다 작거나 같을 확률을 의미합니다. 는 확률밀도함수 아래의 영역 중에서 값 왼쪽 영역의 크기를 의미합니다. 그림으로 누적분포함수와 확률밀도함수의 관계를 표현하면 그림 10.10과 같습니다.

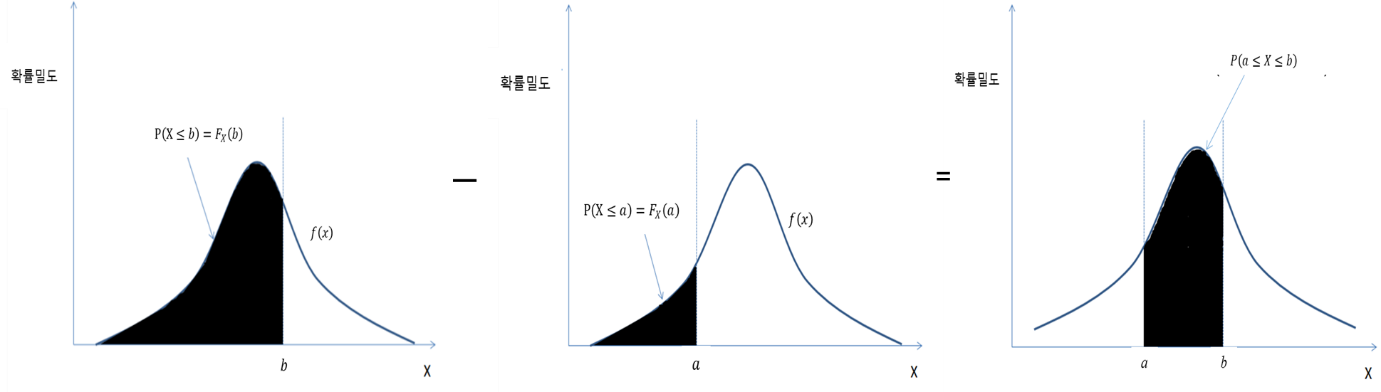


**그림 10.11 확률밀도함수와 누적분포함수와의 관계**

그렇다면, 연속변수 X가 특정 구간 (예, 사이의 값을 취할 확률(즉, )을 누적분포함수를 이용해서 어떻게 구할 수 있을까요? 이는 아래와 같이 계산됩니다.

위에서 는 이고 는 입니다. 따라서 가 됩니다.

이를 해당 변수의 확률밀도함수를 이용해서 그래프로 표현하면 그림 10.11과 같습니다.



**그림 10.12 의 예**

이번에는 이산변수의 누적분포함수에 대해서 살펴보겠습니다. 이산변수의 누적분포함수도 연속변수와 동일한 방식으로 정의됩니다. 이산변수 X에 대한 누적분포함수는 다음과 같습니다.

위 식에서 중요한 것이 부분입니다. 즉 이산변수이지만 누적분포함수는 모든 실수에 대해서 정의된다는 것입니다.

예를 들어 보도록 하겠습니다. 두 개의 동전을 동시에 던지는 시행에 대해서 변수 X를 앞면의 수라고 정의하는 경우, X에 대한 누적분포함수를 찾아보도록 하겠습니다. X가 취할 수 있는 값들은 {0, 1, 2}이고, 각 값을 취할 확률 (즉, 확률질량함수의 값)은 아래와 같습니다.

X의 누적분포함수를 구하기 위해서 X가 취할 수 있는 값, 즉, 를 다음과 같은 구간들로 구분해 보겠습니다. 인 경우, 변수 X는 0보다 작은 값을 취할 수 없기 때문에 . 따라서, 아래의 결과를 갖습니다.

반대로 인 경우에는

가 성립됩니다. 에는 가 모두 포함되기 때문에, 이 됩니다. 그렇기 때문에 2 이상인 어떠한 값에 대해서도 가 성립합니다.

인 경우에는,

인 경우에는,

가 됩니다.

이를 정리하면 아래와 같습니다. 아래는 이산변수 X의 누적분포함수가 됩니다.

위와 같이, 이산변수의 누적분포함수라고 할지라도 모든 실수에 대해서 구하는 것이 필요합니다.

### 변수의 평균, 분산, 공분산

데이터 분석의 많은 경우, 변수가 갖는 분포의 특성을 파악하는 것이 중요합니다. 그 중에서도 분포의 위치와 형태를 파악하는 것이 중요합니다. 분포의 위치와 형태를 결정하는데 있어 가장 큰 역할을 하는 것이 해당 변수의 평균[[25]](#footnote-26)과 분산입니다.

이산변수의 평균은 아래와 같이 정의됩니다.

에서 E는 Expected value를 의미하고, 는 이산변수 X가 취할 수 있는 값을 의미합니다. 는 변수 X의 확률질량함수입니다.

구체적인 예를 들어 보겠습니다. 변수 X를 두개의 동전을 던지는 경우에 나오는 앞면의 수라고 하는 경우에 X가 취할 수 있는 값들은 {0, 1, 2}고 각 값을 취할 확률은 아래와 같습니다.

따라서 변수 X의 평균은 아래와 같습니다.

연속변수 X에 대한 평균은 아래와 같이 정의됩니다.

는 변수 X의 확률밀도함수입니다.

구체적인 예를 들어 보겠습니다. 연속변수 X가 아래와 같은 확률밀도함수를 갖는다고 가정합니다.[[26]](#footnote-27)

연속변수의 평균에 대한 공식을 사용하면 아래와 같이 표현됩니다.

X에 대한 함수인 에 대해서 의 평균은 다음과 같이 정의됩니다.

예를 들어서, 이라고 가정합니다. 이러한 경우, 의 평균은 다음과 같이 표현됩니다.

변수의 평균과 관련된 주요한 공식에는 아래와 같은 것이 있습니다.

변수 X와 상수 c, d에 대해서 아래 공식이 만족합니다.

|  |  |
| --- | --- |
| 공식 | 예 |
|  |  |
|  |  |
|  |  |

참고로 가 됩니다. 이는 변수의 경우 자기 자신을 조건으로 했을 때 상수로 간주되기 때문에 그렇습니다.

이번에는 변수 X의 분산에 대해서 알아보겠습니다. 변수 X의 분산()은 아래와 같이 정의됩니다.[[27]](#footnote-28)

여기서 는 X의 평균입니다. 를 로 표현하고 위 식의 오른쪽 부분을 풀어서 기술해 보겠습니다. 는 구체적인 값을 갖으므로 상수가 됩니다.

평균에 대한 공식을 사용하면 이 됩니다. 여기에서 이므로

가 됩니다. 따라서 변수 X의 분산은 아래와 같이 표기될 수 있습니다.

여기서 는 X가 이산변수인 경우에는

이 되고, X가 연속변수인 경우에는 아래와 같이 됩니다.

분산 관련해서는 상수 c, d에 대해서 아래 식이 만족합니다.

예를 들어 는 이 됩니다.

그리고 의 경우는 X가 상수로 간주되어 0이 됩니다.

두 변수 X와 Y의 공분산은 로 표현되고, 이는 두 변수의 값이 함께 변하는 정도를 나타내며, 다음과 같이 정의됩니다.

여기에서 는 를, 는 를 의미합니다. 위 식에서 오른쪽 항은 아래와 같이 표현될 수 있습니다.

여기서 는 평균의 정의에 의해 다음과 같이 표현됩니다.

여기서 는 변수 X와 Y의 결합확률밀도함수가 됩니다. 이에 대해서는 잠시 후에 조금 더 자세히 다루도록 하겠습니다.

두 변수 X, Y가 서로 독립인 경우에는 가 되어 공분산은 0이 됩니다 (즉, ).

변수 X, Y와 상수 에 대해 공분산 관련 주요 공식은 아래와 같습니다.

### 모멘트 (Moment)

모멘트는 확률분포의 형태를 나타낼 때 사용되는 지표라고 생각할 수 있습니다. 확률분포를 나타내는 확률밀도함수 에 대한 n차 모멘트는 다음과 같이 정의됩니다.[[28]](#footnote-29)

위 식에서 이고 인 경우, 우리는 아래와 같은 모멘트를 갖습니다.

이를 0에 대한 혹은 0을 중심으로한 1차 모멘트라고 표현하며, 이는 즉, 해당 분포의 평균을 의미합니다.

이번에는 이고 인 경우를 살펴보겠습니다. 이는 아래와 같이 됩니다.

이러한 모멘트를 평균에 대한 혹은 평균을 중심으로한 2차 모멘트라고 합니다. 이는 가 되므로, 평균에 대한 2차 모멘트는 해당 분포의 분산을 나타냅니다.

확률 분포의 형태를 결정하는 지표로 3차, 4차 모멘트와 관련된 것이 있습니다. 3차 모멘트와 관련된 것이 왜도 (skewness, 기울어진 정도)이고, 4차 모멘트와 관련된 것이 첨도(kurtosis, 뾰족한 정도)입니다. 왜도와 첨도는 표준화된 모멘트로 표현됩니다. 즉, 평균을 빼고 표준편차로 나눈 형태를 띄게 됩니다.

표준화된 3차 모멘트는 아래와 같이 정의되고, 이를 왜도라고 합니다.

표준화된 4차 모멘트는 아래와 같이 정의되고, 이를 첨도하고 합니다.

### 결합확률분포 (Joint probability distribution)

지금까지는 하나의 변수가 갖는 확률분포 (즉, 확률질량함수 또는 확률밀도함수)에 대해서 살펴보았습니다. 데이터 분석을 수행하다 보면 여러 개의 변수들을 동시에 다루게 됩니다. 여러 개의 변수들에 대한 확률분포를 결합확률분포라고 합니다. 여기서는 설명을 위해 제일 간단한 경우인 변수가 2개인 경우에 대해서 살펴보도록 하겠습니다.

#### 두 개의 이산변수

먼저 복수의 이산변수에 대한 결합확률분포를 나타내는 결합확률질량함수 (joint probability mass function)에 대해서 알아보겠습니다. 두 이산변수 X, Y에 대한 결합확률질량함수는 아래와 같이 표현됩니다.

위 식은 변수 X가 의 값을 갖고 **동시에** 변수 Y가 의 값을 갖을 확률을 의미합니다.

만약 변수 X가 취할수 있는 값들의 집합을 이라고 하고, 변수 Y가 취할 수 있는 값들의 집합을 라고 하는 경우, X와 Y가 동시에 취할 수 있는 값들의 집합은 다음과 같이 표현됩니다.

그리고 X와 Y가 취할 수 있는 모든 값에 대해서 아래 식이 성립합니다.

이 번에는 주변확률분포 (marginal probability distribution)에 대해서 알아보겠습니다. 주변확률분포는 다른 변수가 어떠한 값을 취하느냐와 상관없이 하나의 변수가 특정 값을 취할 확률을 나타내는 분포를 의미합니다. 이산변수의 경우에는 주변확률질량함수(marginal probability mass function)를 이용해서 표현됩니다.

두 이산변수 X, Y의 결합확률질량함수를 이용해서 두 변수 중 한 변수의 확률질량함수를 구할 수 있습니다. 이렇게 결합확률질량함수를 이용해서 도출되는 한 변수의 확률질량함수를 주변확률질량함수라고 합니다.

두 변수 X, Y에 대해서 X의 주변확률질량함수는 다음과 같이 정의됩니다.

이는 변수 Y가 취할 수 있는 모든 값에 대해서 X와 Y의 결합확률질량함수의 값을 더한 것입니다.

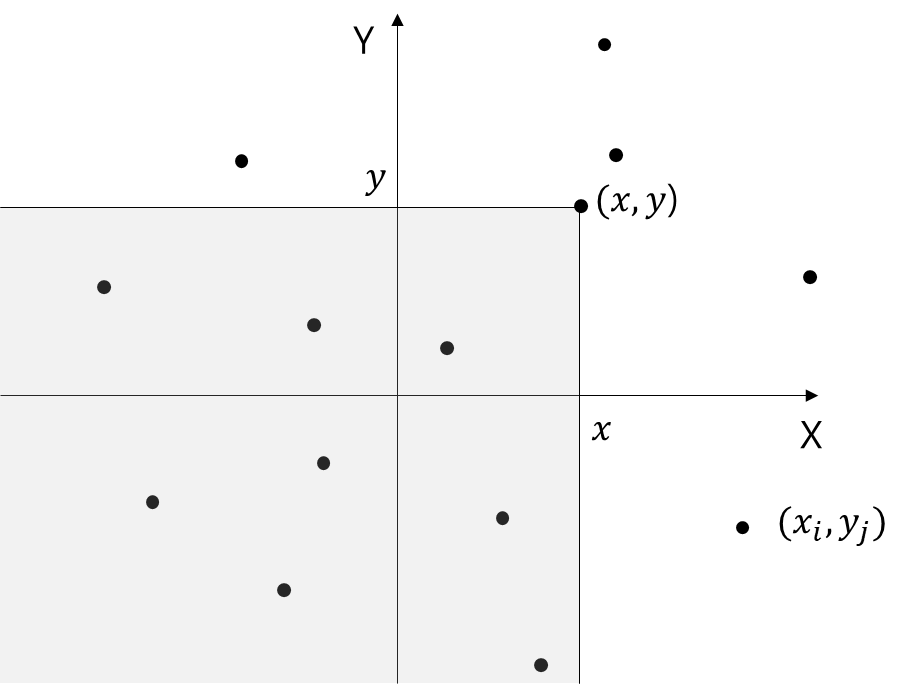
동일한 방식으로 Y의 주변확률질량함수를 구할 수 있습니다.

예) 두 이산변수 X, Y의 결합 PMF가 다음과 같을 때, 각 변수의 주변 PMF를 구하시오 (see Pishiro-Nik p.221)

이번에는 두 변수 X, Y에 대한 결합누적분포함수를 살펴보겠습니다. 이산변수 X의 누적분포함수가 다음과 같이 정의 되는 경우에

두 이산변수 X, Y의 결합누적분포함수는 다음과 같이 정의된다.

이는 변수 X의 값이 이하이면서 동시에 Y의 값이 이하일 확률을 의미합니다. 이를 그림으로 표현하면 그림 10.13과 같습니다.



**그림 10.13 에 해당하는 영역**

결합 CDF에 대해서 주변 CDF는 다음과 같이 정의됩니다.

Y의 주변 CDF도 동일하게 정의됩니다.

Conditional PMF

서로 독립인 두 이산변수들

Conditional Expectation

이산변수

연습문제 (책 볼 것)

연속변수

#### 두 개의 연속변수

Two Continuous random variables

Joint pdf

여기에서

를 joint pdf라고 함

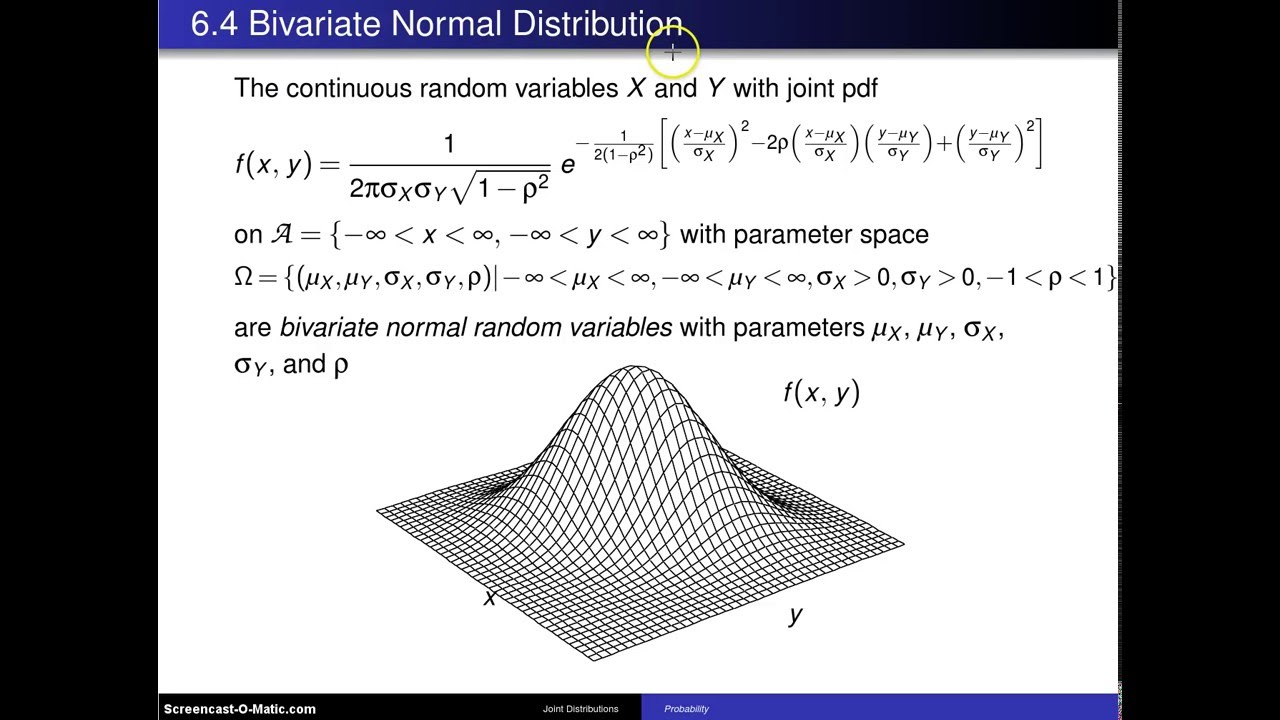
서로 독립인 두 연속변수

The joint probability density function (joint pdf) of X and Y is a function f(x, y) giving the probability density at (x, y).

Joint normal distribution

Bivariate normal distribution

<https://en.wikipedia.org/wiki/Multivariate_normal_distribution#Bivariate_case>



독립인 두 연속 변수

# 문서의 벡터화 (Vectorization)

앞 장에서 관측치 (혹은 데이터 포인트)를 벡터로 변환하는 방법과 그 의미에 대해서 살펴보았습니다. 텍스트 데이터를 기계학습 알고리즘을 이용해서 분석을 하기 위해서도 텍스트 데이터를 구성하는 각 관측치를 벡터로 표현해야 합니다. 텍스트 데이터의 경우에는 일반적으로 문서 하나가 하나의 관측치로 간주 됩니다. 즉, 하나의 문서를 하나의 벡터로 표현하는 것이 필요합니다.

일반적인 하나의 관측치는 관측치가 갖고 있는 특성 정보(features)를 사용해서 벡터로 표현합니다. 그렇다면, 문서의 경우에는 어떠한 정보를 특성 정보로 사용할 수 있을까요? 즉 문서가 가지고 있는 정보 중에서 문서의 고유한 특성을 잘 반영하는 정보가 무엇이 있을까요? 대표적인 것이 바로 단어입니다 (한글은 형태소가 될 수 있습니다). 어떠한 단어들이 얼마만큼 사용되었느냐에 따라서 문서의 특성이 크게 달라지게 됩니다.

일반적으로 문서를 벡터로 변환할 때 문서에서 사용된 모든 단어의 정보를 사용하지 않습니다. 분석에 있어서 중요한 역할을 하는 단어들의 정보만을 특성 정보로 해서 각 문서를 벡터로 표현하게 됩니다. 많은 경우, 전처리 과정을 거쳐 추출된 특정한 단어들만[[29]](#footnote-30)을 사용해서 문서를 벡터로 변환하게 됩니다. 문서의 특성을 잘 나타내는 단어 정보를 사용해서 벡터로 변환하게 되면 해당 벡터는 문서의 특성을 반영하게 됩니다. 따라서 벡터 간의 유사도를 계산하여 문서 간의 유사도를 계산할 수 있게 되는 것입니다.

그렇다면 우리는 문서에서 사용된 단어의 어떠한 정보를 사용해서 문서를 벡터로 표현할 수 있을까요? 기계학습에서 많이 사용되는 방법은 아래 두 가지 입니다.

① 문서에서 사용된 단어들의 빈도 정보 (이를 term frequency라고 합니다. 여기서 term은 단어를 의미합니다.)

② 문서에서 사용된 단어들의 상대적 빈도 정보 (보통 term-frequency\*inverse-document-frequency 값을 사용합니다. 이를 줄여서 TF-IDF라고 합니다.)

## 단어들의 빈도 정보를 사용해서 문서를 벡터로 표현하기

각각의 단어들이 특정 문서에서 몇 번 사용되었는지의 정보를 사용해서 문서를 벡터로 변환하는 방법입니다. 이러한 방법을 단어가방모형 (Bag-of-Words Model, 이하 BoW)이라고 합니다. 문서를 단어들이 (순서없이) 들어있는 가방으로 간주하는 것입니다. 즉, BoW 방법은 문서에서 사용된 단어들의 순서를 고려않고 사용 빈도만을 고려합니다.[[30]](#footnote-31) 그래도 많은 경우 문서의 특성을 잘 반영합니다. BoW 방법을 사용해서 문서를 어떻게 벡터로 변환할 수 있는지를 설명하기 위해 그림 8.1의 예제 텍스트 데이터를 사용하도록 하겠습니다. 해당 텍스트 데이터는 Doc 1, Doc 2, Doc 3의 세 개의 문서들로 구성되어 있습니다. 그리고 각 문서들은 여러 개의 단어들로 구성되어 있습니다 (해당 단어들은 전처리 과정을 거친 단어들이라고 가정합니다).

**Doc 1: ‘banana apple apple orange’**

**Doc 2: ‘apple carrot eggplant carrot’**

**Doc 3: ‘banana mango orange orange’**

**그림 8.1 예제 텍스트 데이터**

각 문서를 벡터를 변환하기 위해서 먼저 전체 텍스트 데이터, 즉 말뭉치에서 사용된 (중복없는) 단어들(이를 vocabulary라고 합니다)이 무엇인지 파악합니다. 그리고 전체 단어들을 알파벳 순서로 배열합니다 (한글은 가나다 순이 됩니다). 위의 세 개의 문서에서 사용된 전체 단어들을 알파벳 순서로 배열하면 아래와 같습니다.

apple, banana, carrot, eggplant, mango, orange

위의 단어들이 각 문서를 벡터로 변환할 때 사용되는 특성 정보(features)가 되는 것입니다. 각 문서에 대해서 각 단어들이 몇 번씩 사용되었는지를 파악합니다. 이를 정리하면 그림 8.2와 같습니다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **apple** | **banana** | **carrot** | **eggplant** | **mango** | **orange** |
| **Doc 1** | 2 | 1 | 0 | 0 | 0 | 1 |
| **Doc 2** | 1 | 0 | 2 | 1 | 0 | 0 |
| **Doc 3** | 0 | 1 | 0 | 0 | 1 | 2 |

**그림 8.2 각 문서에 대한 단어들의 빈도**

첫 번째 문서 (Doc 1)의 경우, apple이 2번, banana가 1번 carrot, eggplant, mango 는 0번, orange는 1번 사용되었다라는 것을 의미합니다. 그림 8.2와 같은 행렬을 문서-단어 행렬 (Document-Term Matrix, DTM)이라고 합니다. 해당 행렬은 3x6 행렬 즉, 행의 수가 3이고 열의 수가 6인 행렬이 되는데 여기서 행의 수는 문서의 수를 의미하고 열의 수는 각 문서를 벡터로 표현할 때 사용된 단어의 수를 의미합니다. 즉, DTM의 각 행이 각 문서에 대한 벡터가 되는 것입니다. Doc1=(2,1,0,0,0,1), Doc2=(1,0,2,1,0,0), Doc3=(0,1,0,0,1,2)가 됩니다. 여러분이 보는 것 처럼 각 문서가 갖는 원소의 수는 동일하고, 이는 (전처리가 끝난) 텍스트 데이터에서 사용된 단어의 수가 됩니다. 위의 예에서는 전처리 결과물 얻어진 텍스트 데이터에 여섯 개의 단어가 있기 때문에 각 문서가 6차원의 벡터로 표현되었습니다. 원소의 수가 6개이기 때문에 각 벡터는 6차원 공간에 존재하는 하나의 점이 됩니다. 그리고 해당 점의 위치는 각 단어가 해당 문서에서 얼마나 자주 사용되었는지에 따라서 달라집니다. 동일한 단어가 비슷한 정도로 사용된 문서에 대한 벡터 간 유사도가 높아지게 됩니다. 왜냐하면 그러한 문서들에 대한 벡터 간의 위치가 더 가깝기 때문입니다.

각 문서를 벡터로 변환한 후에는 앞에서 배운 여러가지 방법을 사용해서 벡터 간의 유사도 (즉, 문서 간의 유사도)를 계산할 수 있습니다. 여기서는 유클리디안 거리와 코사인 유사도를 계산해 보겠습니다. 관련 코드는 doc\_vectorization\_toy.ipynb 파일을 참고하세요

numpy를 이용해서 아래와 같이 세 개의 벡터를 생성합니다 (지금은 벡터의 원소를 직접 입력하지만 잠시 후에는 sklearn 모듈을 사용해서 변환하는 방법을 알아볼 것입니다).

import numpy as np

Doc1 = np.array([2, 1, 0, 0, 0, 1])

Doc2 = np.array([1, 0, 2, 1, 0, 0])

Doc3 = np.array([0, 1, 0, 0, 1, 2])

문서1과 문서2 벡터 사이의 유클리디안 거리는 아래와 같이 norm() 함수를 이용해서 계산할 수 있습니다.

print(np.linalg.norm(Doc2-Doc1))

2.8284271247461903

문서1과 문서2 사이의 코사인 유사도는 다음과 같이 내적 공식을 이용해서 계산할 수 있습니다.

np.dot(Doc1, Doc2)/(np.linalg.norm(Doc1)\*np.linalg.norm(Doc2))

0.33333333333333337

**단어의 빈도 정보 (term-frequency)를 이용한 방법의 한계**

단어의 빈도 정보를 이용해서 문서를 벡터로 표현하는 방법은 한 가지 한계가 있습니다. 바로 단어들이 각 문서에 대해 갖는 상대적 중요성을 반영하지 못한다는 것입니다. 이는 다르게 표현하면 각 단어의 정보가 문서의 고유한 특성을 잘 나타내지 못한다는 것을 의미합니다. 설명을 위해서 그림 8.3의 예를 보도록 하겠습니다. 각 셀의 값은 단어의 빈도를 의미합니다.

|  |  |  |
| --- | --- | --- |
|  | Word 1 | Word 2 |
| Doc 1 | 10 | 0 |
| Doc 2 | 10 | 10 |

**그림 8.3 예제 텍스트 데이터**

위와 같은 DTM이 있다고 할 때 Word 1과 Word 2 중 어떠한 단어가 Doc 2의 고유한 특성을 더 잘 반영할까요? Doc 2 경우, Word 1과 Word 2가 모두 10번 같은 횟수로 사용되었다는 것을 알 수 있습니다. 하지만, 데이터에 존재하는 다른 문서에서 각 단어들이 사용된 정도까지 고려하면 상대적으로 Word 2가 Doc 2의 고유한 특성을 더 잘 반영한다는 것을 알 수 있습니다. 왜냐하면 Word 1은 Doc 1에서도 열 번 사용된 반면, Word 2는 Doc 1에서 사용되지 않았기 때문입니다. 즉, 각 단어들이 각 문서의 고유한 특성을 얼마나 잘 표현하는지를 좀 더 정확하게 파악하기 위해서는 단어들이 해당 문서에서 (다른 문서들에서 보다) 상대적으로 얼마나 더 많이 사용되었는지를 알아야 합니다. 예를 들어, 특정 문서에서는 많이 사용되고 다른 문서에서는 사용된 정도가 적은 단어는 해당 문서의 고유한 특성을 잘 반영하는 것입니다. 이를 위해서 우리는 TF-IDF (Term frequency – Inverse document frequency) 지표를 사용합니다.

## TF-IDF (Term frequency – Inverse document frequency) 정보 사용하기

각 단어들이 갖는 TF-IDF 값은 TF의 값에 IDF 값을 곱해서 계산이 됩니다 (즉, TF\*IDF). 여기서, TF (term frequency)는 해당 단어가 해당 문서에서 사용된 횟수가 됩니다. 여기에 추가적으로 IDF 값을 고려해서 TF-IDF를 계산하게 됩니다. IDF (Inverse document frequency)는 해당 단어가 얼마나 많은 문서에서 사용되지 않았는지를 의미합니다. IDF는 DF의 역수에 해당합니다 (즉, 1/DF). DF (document frequency)는 해당 단어가 사용된 텍스트 데이터에 존재하는 문서의 수를 의미합니다. 즉, 단어가 사용된 문서의 수가 작을수록 (즉, DF의 값이 작을수록) IDF 값이 커집니다. 해당 문서에서 많이 사용되고 (즉, TF값의 크고), 사용된 문서의 수가 작을수록 (즉, IDF 값이 크고) TF-IDF 값이 커지게 됩니다. 즉, 해당 문서에 대한 상대적 빈도값이 커지게 되는 것입니다. 이는 해당 단어가 해당 문서에서 상대적으로 중요한 역할을 한다는 것을 의미합니다 (혹은 해당 문서의 고유한 특성을 잘 반영한다는 것을 의미합니다).

아래의 데이터에 대해서 직접 TF-IDF를 계산해 보겠습니다. TF의 값은 아래 행렬에 표시되어 있습니다.

|  |  |  |
| --- | --- | --- |
|  | Word 1 | Word 2 |
| Doc 1 | 10 | 0 |
| Doc 2 | 10 | 10 |

**그림 8.4 TF 정보**

이번에는 각 단어가 각 문서에 대해 갖는 DF 값을 계산해 보겠습니다. DF는 해당 단어가 사용된 문서의 수를 의미합니다. DF 값은 아래와 같습니다. 첫 번째 셀의 값인 2은 Word 1이 텍스트 데이터에 존재하는 두 개의 문서 (즉, Doc1과 Doc2)에서 사용된 것을 의미합니다. 네 번째 셀의 값 1은 Word 2가 하나의 문서 (Doc 2)에서만 사용되었다라는 것을 의미합니다.

|  |  |  |
| --- | --- | --- |
|  | Word 1 | Word 2 |
| Doc 1 | 2 | 2 |
| Doc 2 | 2 | 1 |

**그림 8.5 DF 정보**

이번에는 DF 정보를 이용해서 IDF 값을 계산해 보겠습니다. 여기서는 설명을 간단하기 위해 1/DF의 값으로 IDF의 값을 계산해 보겠습니다.[[31]](#footnote-32) IDF 값은 아래와 같습니다.

|  |  |  |
| --- | --- | --- |
|  | Word 1 | Word 2 |
| Doc 1 | ½ | ½ |
| Doc 2 | ½ | 1 |

**그림 8.6 IDF 정보**

TF 값과 IDF 값을 곱하여 아래와 같이 TF-IDF 값을 구합니다.

|  |  |  |
| --- | --- | --- |
|  | Word 1 | Word 2 |
| Doc 1 | 10\*½ =5 | 10\*½=5 |
| Doc 2 | 10\*½ =5 | 10\*1=10 |

**그림 8.7 TF-IDF 정보**

이러한 TF-IDF 정보를 이용해서 각 문서를 벡터로 표현하면 Doc1 = (5, 5), Doc2 = (5,10)으로 표현됩니다. 앞에서 얘기했던 것 처럼 Word 2가 Doc 2에 대해서 갖는 TF-IDF 값이 큰 것을 알 수 있습니다. 이는 해당 단어가 Doc 2에서 상대적으로 더 중요한 역할을 한 것을 알 수 있습니다 (혹은 Doc 2의 고유한 특성을 더 잘 나타낸다라고 생각할 수 있습니다).

## sklearn을 이용해서 문서를 벡터로 변환하기

파이썬에서는 기계학습에 사용되는 sklearn[[32]](#footnote-33) 모듈을 사용해서 문서를 벡터로 변환합니다. 문서를 벡터로 변환하기 위해서는 sklearn에서 제공되는 아래 두 가지 클래스를 사용합니다.

* CountVectorizer 클래스: 출현 빈도 (즉, term frequency)를 이용해서 문서를 벡터로 표현하고자 하는 경우에 사용합니다.
* TfidfVectorizer 클래스: TF-IDF 값을 이용해서 문서를 벡터로 표현하고자 하는 경우에 사용합니다.

각 클래스를 사용해서 문서를 벡터로 변환하는 코드는 doc\_vectorization\_example1.ipynb 파일에 저장되어 있습니다.

여기서는 아래의 예제 텍스트 데이터를 사용합니다. 해당 텍스트 데이터는 네 개의 문서들로 구성되어 있고, 각 문서는 전처리를 거친 단어들로 구성되어 있다고 가정합니다.

Doc 1 = 'banana apple apple eggplant',

Doc 2 = 'orange carrot banana eggplant',

Doc 3 = 'apple carrot banana banana',

Doc 4 = 'orange banana grape'

**그림 8.8 예제 텍스트 데이터**

CounterVectorizer와 TfidfVectorizer 클래스를 이용해서 각 문서를 벡터로 표현하기 위해서는 텍스트 데이터를 아래와 같이 각 문서를 하나의 문자열 값의 원소로 갖는 리스트 데이터로 저장합니다.

TEXT = [

'banana apple apple eggplant',

'orange carrot banana eggplant',

'apple carrot banana banana',

'orange banana grape'

]

### 빈도 정보 (Term frequency)를 사용하여 벡터로 표현하기

TEXT 변수에 저장되어 있는 각 문서를 단어 사용 빈도 정보를 이용해서 벡터로 표현하기 위해서는 CountVectorizer 클래스를 사용합니다. 이를 위해 다음과 같이 해당 클래스를 임포트합니다.

from sklearn.feature\_extraction.text import CountVectorizer

CountVectorizer 클래스의 함수를 사용해서 문서들을 벡터로 표현하기 위해서 일단 아래와 같이 해당 클래스의 생성자 함수를 이용해서 객체를 만듭니다.

tf\_vectorizer = CountVectorizer(min\_df=2, max\_df=0.8, ngram\_range=(1,1))

CountVectorizer() 생성자 함수는 주요한 파라미터가 세 개 있습니다. min\_df, max\_df, ngram\_range가 그것입니다. min\_df와 max\_df 단어가 사용된 문서의 수를 기준으로 문서를 벡터로 표현할 때 사용하고자하는 단어들을 추가적으로 선별하고자 하는 목적으로 사용합니다. min\_df는 minimum document frequency를 의미합니다. min\_df로 지정된 값보다 더 적은 문서에서 사용된 단어들은 문서를 벡터로 표현할 때 고려되지 않습니다. min\_df는 0 이상의 정수 또는 0과 1사이 소수(float)를 값으로 입력 받습니다. 0 이상의 정수인 경우에는 절대적 문서 숫자를 의미하고 0과 1의 소수인 경우에는 비율(ratio)을 의미합니다. 예를 들어서 min\_df = 2로 지정하면 전체 문서 중에서 2개 미만의 문서에서 사용된 단어들은 벡터화 작업에서 제외합니다 (즉, 한 개의 문서에서만 사용된 단어들은 제외됩니다). 만약, min\_df = 0.1로 지정하면 전체 문서 중에서 10%에 해당하는 문서 미만에서 사용된 단어들은 제외됩니다.

max\_df는 maximum document frequency 을 의미합니다. max\_df로 지정된 값보다 더 많은 문서에서 사용된 단어들 역시 벡터 변환에 사용되지 않습니다. max\_df 또한 0 이상의 정수 또는 0과 1사이의 소수를 값으로 입력 받을 수 있습니다. max\_df = 100 이라고 입력하면 101개 이상의 문서에서 사용된 단어는 벡터 변환에서 제외됩니다. max\_df = 0.8 이라고 한다면 전체 문서 중에서 80% 이상의 문서에서 사용된 단어는 벡터 변환에서 제외 됩니다.

ngram\_range는 벡터로 변환할 때 사용하고자 하는 n-gram의 범위를 지정하기 위해서 사용합니다. 즉 해당 파라미터의 값을 통해서 unigram을 사용할 것인지 bigram 혹은 trigram까지 사용할 것인지를 결정하게 됩니다. 만약, ngram\_range=(1,2) 로 지정한다면 1-gram (즉, unigram)과 2-gram (즉, bigram) 단어들을 벡터 변환에 사용하겠다는 뜻입니다. ngram\_range=(1,1)로 지정하면 unigram 만 사용한다는 것이고, ngram\_range=(2,2) 는 bigram 만을 사용한다는 것입니다. 만약 여러분의 텍스트 데이터에 자주 사용되는 혹은 중요한 역할을 하는 bigram 표현이 있다면 bigram도 포함시키는 것을 고려해 볼 수 있습니다. 하지만, ngram\_range를 통해서는 특정한 기준을 만족하는 bigram만을 포함시킬 수 없다는 단점이 있습니다. 전처리 결과로 나온 단어들로부터 얻을 수 있는 모든 bigram을 포함하게 됩니다.

우리는 이 생성자 함수를 이용해서 tf\_vectorizer라는 이름의 객체를 만들고, 해당 객체를 통해 CouterVectorizer 클래스에서 제공되는 fit\_transform() 함수를 사용해서 문서들을 벡터로 변환합니다. 텍스트 데이터를 저장하고 있는 TEXT 변수를 해당 함수의 인자로 입력합니다.

tf\_features = tf\_vectorizer.fit\_transform(TEXT)

각 문서를 벡터로 변환한 결과가 tf\_features라는 변수에 저장되었습니다. 즉, 전체 데이터에 대한 DTM 정보가 tf\_features 변수에 저장되어 있습니다. print() 함수를 이용해서 tf\_features의 내용을 출력해 보겠습니다.

print(tf\_features)

아래와 같은 결과가 출력 됩니다.

(0, 0) 2

(0, 2) 1

(1, 2) 1

(1, 3) 1

(1, 1) 1

(2, 0) 1

(2, 1) 1

(3, 3) 1

전체 데이터에서 사용된 단어가 많은 경우, 많은 단어들이 DTM에서 갖는 값이 0입니다. 공간의 절약을 위해서 0인 값들은 보통 생략을 하고 저장합니다. tf\_features에는 셀의 값이 0이 아닌 단어들에 대한 결과만 저장되어 있습니다. 위의 결과가 의미하는 것이 무엇인지 살펴보겠습니다. 첫번째 행의 값은 아래와 같습니다.

(0, 0) 2

위의 결과에서 튜플의 첫번째 숫자 (0)은 문서 ID입니다. 즉, 첫 번째 문서('banana apple apple eggplant')를 의미합니다. 그리고 tuple의 두 번째 숫자는 단어의 ID 입니다. 그 값이 0이므로 첫 번째 단어를 의미합니다 (1번째 단어가 무엇인지 우리는 아직 알지 못합니다).[[33]](#footnote-34) 그리고 튜플 다음에 나오는 숫자 2는 해당 단어가 해당 문서에서 사용된 빈도 (frequency)를 의미합니다. 즉 첫 번째 단어 (단어 0)는 첫 번째 문서에서 두 번 사용되었다는 뜻입니다.

이러한 저장 방식은 컴퓨터 관점에서는 효율적이지만, 사람들이 보기에는 그 다지 효과적이지는 않습니다. 우리가 친숙한 0을 포함한 DTM으로 표현해 보겠습니다. 이를 위해서는 다음과 같이 todense() 함수를 사용합니다.

features = tf\_features.todense()

이 결과를 화면에 출력해 보면

features

다음과 같은 결과가 나옵니다.

array([[2, 0, 1, 0],

[0, 1, 1, 1],

[1, 1, 0, 0],

[0, 0, 0, 1]], dtype=int64)

사람이 해석하기에 좀 더 용이합니다. 위의 결과는 전체 문서에 대한 DTM을 의미하므로 각 행이 각 문서에 대한 벡터가 됩니다. 예를 들어, 첫 번째 행은 첫번째 문서에 대한 벡터가 되는 것입니다. 그리고 열은 벡터 변환에 사용된 단어들을 의미합니다. TEXT 변수에 사용된 전체 단어는 여섯 개이지만 우리가 CountVectorizer 클래스 생성자 함수의 min\_df 파라미터의 값을 2로 설정하여 1개의 문서에서만 사용된 단어들이 벡터 변환에서 제외 되었습니다 (grape가 하나의 문서에서만 사용되었습니다). 뿐만 아니라 max\_df = 0.8로 지정함에 따라 80%가 넘는 문서에서 사용된 단어도 제외 되었습니다 (모든 문서에서 사용된 banana가 제외 되었습니다). 그래서 각 문서가 (banana와 grape를 제외한) 총 4개의 단어 정보에 대해서만 벡터로 표현 되었습니다. 첫 번째 문서에 대한 벡터는 다음과 같습니다. 즉, 첫 번째 단어가 두번 사용되었고, 두 번째 단어와 네 번째 단어는 0번, 그리고 세 번째 단어는 한 번 사용되었다는 것을 알 수 있습니다.

[2, 0, 1, 0]

그러면 이제는 각 단어가 무엇인지 확인해 보겠습니다. 이를 위해서는 CounterVectorizer 클래스에서 제공되는 get\_feature\_names\_out() 함수를 사용합니다.

feature\_names = tf\_vectorizer.get\_feature\_names\_out()

get\_feature\_names() 함수는 벡터 표현에 사용된 단어들의 이름을 반환합니다. 여기서 feature는 단어를 의미합니다. 다음과 같이 feature\_names를 화면에 출력해 보겠습니다.

print(feature\_names)

그 결과는 아래와 같습니다.

['apple', 'carrot', 'eggplant', 'orange']

즉, 첫 번째 단어는 apple (즉, index = 0), 두 번째 단어는 corrot (즉, index = 1) 이라는 것을 알 수 있습니다.

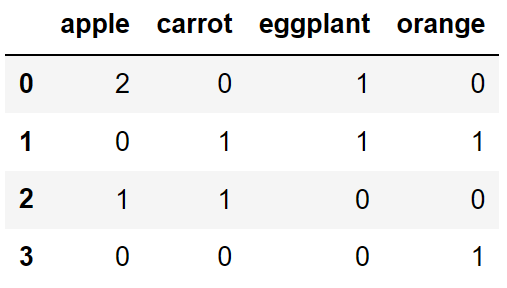
DTM 정보가 저장된 features 변수와 단어들의 정보를 담고 있는 feature\_names 변수를 사용해서 좀 더 보기 좋게 DTM을 화면에 출력해 보겠습니다. 다음과 같이 pandas의 DataFrame 형태로 표현합니다.

import pandas as pd

df = pd.DataFrame(data=features, columns=feature\_names)

df

DataFrame()은 pandas에서 제공되는 DataFrame 클래스의 생성자함수입니다. 이 생성자함수는 data 파라미터에 입력되는 데이터와 columns 파라미터에 입력되는 헤더 정보를 이용해서 DataFrame 형태의 데이터를 반환하는 역할을 합니다 (DataFrame 형태의 데이터는 엑셀의 테이블 형태와 비슷합니다). 위 결과는 아래와 같습니다.



**그림 8.9 DTM 출력 결과**

DTM 정보를 담고있는 features 변수를 이용해서 우리는 각 문서의 벡터를 추출할 수 있습니다. 이를 위해서 다음과 같이 인덱싱 방법을 사용합니다. 첫 번째 문서에 대한 벡터는 DTM의 첫 번째 행이므로 features[0]을, 두 번째 문서의 벡터는 features[1]을 사용합니다. 추출된 각 문서를 나타내는 벡터들을 이용해서 문서간의 유사도를 구할 수 있습니다. 유클리디안 거리와 코사인 유사도는 다음과 같이 구합니다.

# 첫 번째 문서와 두 번째 문서 간 유클리디안 거리

print(np.linalg.norm(features[1]-features[0]))

2.449489742783178

# 첫 번째 문서와 두 번째 문서 간 코사인 유사도

print(np.dot(features[0],features[1])/(np.linalg.norm(features[0])\*np.linalg.norm(features[1])))

0.2581988897471611

### TF-IDF 정보를 사용해서 벡터로 표현하기

이번에는 TfidfVectorizer 클래스를 사용해서 TF-IDF 기반의 벡터로 변환해 보도록 하겠습니다. 이를 위해 아래와 같이 해당 클래스를 임포트하고, 생성자함수를 이용해서 객체를 만듭니다. TfidfVectorizer() 생성자함수도 CountVectorizer() 생성자함수와 동일한 파라미터들을 갖습니다. 여기서도 각 파라미터의 값을 동일하게 설정합니다 (즉, min\_df=2, max\_df=0.8, ngram\_range=(1, 1)).

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf\_vectorizer = TfidfVectorizer()

TF-IDF 기반의 DTM을 얻기 위해서 TfidfVectorizer 클래스의 fit\_transform() 함수를 사용합니다. TEXT 변수를 해당 함수의 인자로 입력합니다.

tfidf\_features = tfidf\_vectorizer.fit\_transform(TEXT)

TF-IDF 기반의 DTM 정보가 tfidf\_features 변수에 저장되어 있습니다. 이를 print() 함수를 이용해서 출력해 보겠습니다.

print(tfidf\_features)

(0, 2) 0.4472135954999579

(0, 0) 0.8944271909999159

(1, 1) 0.5773502691896257

(1, 3) 0.5773502691896257

(1, 2) 0.5773502691896257

(2, 1) 0.7071067811865475

(2, 0) 0.7071067811865475

(3, 3) 1.0

TF 기반의 DTM 정보를 저장하고 있는 tf\_features 변수와 마찬가지로 tfidf\_features 변수에도 메모리 공간을 절약하기 위해서 TF-IDF의 값이 0이 아닌 값들만 저장되어 있습니다. 첫 번째 행의 결과를 살펴보겠습니다.

(0, 2) 0.4472135954999579

튜플의 첫 번째 원소는 문서의 ID를 두 번째 원소는 단어의 ID를 의미합니다. 튜플 옆의 숫자 (0.4472135954999579)는 해당 단어가 해당 문서에 대해서 갖는 TF-IDF 값입니다. TfidfVectorizer 클래스에서는 IDF 값은 아래 공식을 이용해서 계산됩니다.

는 단어 의 IDF 값이라는 것을 의미합니다. 은 텍스트 데이터에 존재하는 전체 문서의 수를, 는 단어 가 사용된 문서의 수가 됩니다. 우리가 다루고 있는 예제 데이터의 경우, 입니다.

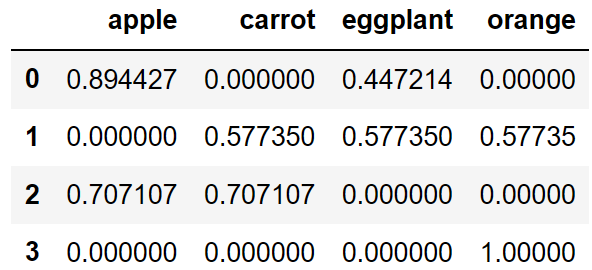
각 단어가 무엇인지 확인하기 위해 이번에도 get\_feature\_names\_out() 함수와 DateFrame()을 사용해서 아래와 같이 DTM을 출력합니다.

tfidf\_feature\_names = tfidf\_vectorizer.get\_feature\_names\_out()

df = pd.DataFrame(data=tfidf\_features, columns=tfidf\_feature\_names)

df

결과는 아래와 같습니다.



**그림 8.10 TF-IDF 기반 DTM**

값이 0인 셀은 TF의 값이 0인 것을 의미합니다. apple 단어가 첫 번째 문서 (문서 0)에 대해서 갖는 TF-IDF 값인 0.894427이 어떻게 계산되었는지 알아보도록 하겠습니다. apple 단어는 텍스트 데이터에 존재하는 두 개의 문서 (문서 0과 문서 2)에서 사용되었으므로 해당 단어의 IDF 값은 아래와 같습니다.

그리고 apple은 문서 0에서 한 번 사용되었으므로 TF = 2입니다. 따라서 TF-IDF = 2\*1.510825 = 3.021651가 됩니다. 하지만, 이 값은 위에 나와 있는 0.894427과 다릅니다. TfidfVectorizer 클래스의 경우는 원값을 그대로 사용하지 않고, 각 문서의 벡터가 원래의 벡터가 아니라 단위 벡터가 되게끔 결과를 출력합니다. 예를 들어 첫 번째 문서인 문서 0의 원래 TF-IDF 벡터는 다음과 같습니다.

문서 0의 원래 TF-IDF 벡터 = (3.021651, 0, 1.510825, 0)

그리고 해당 벡터의 길이 (즉, 원점으로부터의 유클리디안 거리)는 3.37830829이기 때문에 해당 길이로 원래의 벡터를 나누게 되면 아래의 결과를 얻습니다.

문서 0의 단위 벡터 = (3.021651/3.37830829, 0/3.37830829, 1.510825/3.37830829, 0/3.37830829)

= (0.89442725, 0, 0.44721348, 0)

이렇게 얻어진 결과가 위에서 TfidfVectorizer 클래스를 이용해서 얻은 결과와 같은 것을 확인할 수 있습니다. 각 문서의 벡터는 단위 벡터이기 때문에 길이를 구해보면 아래와 같이 1의 값이 나오는 것을 확인할 수 있습니다.[[34]](#footnote-35)

np.linalg.norm(tfidf\_features[0])

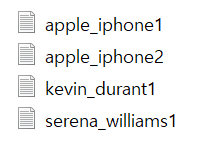
1

TF-IDF의 값이 클수록 해당 단어가 해당 문서의 고유한 특성을 더 잘 나타낸다는 것을 의미합니다. 예를 들어, 첫 번째 문서의 경우 apple 단어의 TF-IDF 값이 제일 크기 때문에, 해당 단어가 첫 번째 문서의 고유한 특성을 제일 잘 나타낸다고 생각할 수 있는 것입니다.

TF-IDF 기반의 벡터들을 이용해서 유클리디안 거리나 코사인 유사도 방법을 사용하여 문서 간의 유사도를 구할 수 있습니다.

### 실제 문서 벡터화 해보기

실제 문서들을 벡터화하는 것에 대한 예제 코드는 doc\_vectorization\_example2.ipynb을 참고하세요. 해당 예제 코드에서는 Docs라고 하는 폴더에 저장이 되어 있는 네 개의 신문기사 (그림 8.11 참고), 즉, 네 개의 문서 데이터를 읽어와서 각 문서를 하나의 벡터로 변환하는 내용을 담고 있습니다.



**그림 8.11 Docs 폴터에 저장되어 있는 네 개의 문서**

그림 8.11을 보면 각 문서를 저장하고 있는 파일의 제목은 해당 문서가 무엇에 대해서 다루고 있는지를 나타냅니다. 첫 번째 문서와 두 번째 문서는 애플 아이폰에 대해서 다루고 있는 서로 다른 신문기사이고, 세 번째 문서는 미국 농구 선수인 케빈 듀란트에 대한 신문기사이고, 네 번째 문서는 미국 테니스 선수인 세레나 윌리엄스에 대한 신문기사입니다.

doc\_vectorization\_example2.ipynb에서는 각 문서를 벡터로 표현하고, 각 벡터 간의 유사도를 구해서 문서 간의 유사도를 파악해 보도록 하겠습니다. 파일 제목에서 알 수 있듯이 첫 번째와 두 번째 문서가 다루는 주제가 동일하기 때문에 두 문서를 나타내는 벡터들 간의 유사도가 상대적으로 크게 나와야 합니다. 정말로 그런 결과가 나오는지 실제로 분석을 해보겠습니다.

여기서는 각 문서를 벡터로 표현하기 전에 전처리 과정을 거쳐서 불용어가 제거된 명사의 단어들을 추출하고 그 결과물을 이용해서 벡터화 작업을 수행하도록 합니다.

일단 먼저 아래와 같이 폴더에 저장되어 있는 각 파일의 내용을 읽어 옵니다.

from os import listdir

from os.path import isfile, join

mypath = './docs/'

# 아래 코드를 사용하여 폴더에 존재하는 파일의 이름만을 저장합니다.

onlyfiles = [f for f in listdir(mypath) if isfile(join(mypath, f))]

total\_docs = []

for file in onlyfiles:

file\_path = mypath+file

with open(file\_path, 'r', encoding='utf8') as f:

content = f.read()

total\_docs.append(content)

위 코드에서는 각 파일의 내용을 read() 함수를 이용해서 하나의 문자열값으로 읽어와서 total\_docs 리스트 변수에 저장합니다. 결과로 total\_docs 변수는 네 개의 원소를 갖고, 각 원소는 각 파일의 내용을 담고 있습니다. 첫 번째 원소는 첫 번째 파일의 내용을, 두 번째 원소는 두 번째 파일의 내용을 저장하고 있습니다.

전처리를 위해서 preprocessing.py 파일에 저장되어 있는 En\_preprocessing() 함수를 사용합니다. 해당 함수는 인자를 두 개를 받습니다. 첫 번째 인자는 전처리를 하고자 하는 원본 텍스트 데이터이고, 두 번째 인자는 사용자 불용어 사전입니다. En\_preprocessing() 함수의 body는 5장에서 다룬 영어 텍스트 전처리 관련 코드로 구성되어 있습니다. 해당 함수는 결과로 불용어가 제거된 명사의 단어들을 리스트 데이터의 형태로 반환합니다

해당 파일을 아래와 같이 임포트 합니다.

import preprocessing

아래와 같이 리스트 데이터 형태로 사용자 불용어 사전을 만듭니다.

stopwords = ['be', 'today', 'yesterday', 'tomorrow']

En\_preprocessing() 함수를 이용해서 각 문서를 전처리하고 그 결과를 아래와 같이 docs\_nouns 리스트 변수에 저장합니다.

docs\_nouns = [preprocessing.En\_preprocessing(doc, stopwords) for doc in total\_docs]

docs\_nouns는 네 개의 원소를 갖고 각 원소는 아래와 같이 각 문서에 대해서 전처리 결과물로 도출된 명사 단어들로 구성된 리스트 데이터입니다.

print(docs\_nouns[0])

['smartphones', 'side', 'desk', 'pocket', 'bag', 'day', 'hour', 'day', 'software', 'iphone', 'phone', 'reveals', 'detail', 'surprise', 'battery', 'anxiety', 'malady', … <이하 생략>

이제 전처리 결과물에 CountVectorizer와 TfidfVectorizer 클래스를 적용해서 각 문서를 벡터로 표현해 보도록 하겠습니다. CountVectorizer와 TfidfVectorizer 클래스를 적용하기 위해서는 각 문서가 하나의 문자열값으로 저장되어 있어야 합니다. 하지만, 위의 경우는 각 문서가 단어들의 리스트 데이터로 저장되어 있습니다. 일단, 이를 아래와 같이 join() 함수를 사용해서 하나의 문자열값으로 변경합니다.

documents\_filtered = []

for doc in docs\_nouns:

document\_filtered =' '.join(doc)

documents\_filtered.append(document\_filtered)

documents\_filtered 변수는 네 개의 원소를 갖는 리스트 변수이고, 각 원소는 각 문서를 나타내는 하나의 문자열값입니다.

먼저 CountVectorizer 클래스를 이용해서 각 문서를 TF 기반의 벡터로 표현해 보도록 하겠습니다. 여기서는 min\_df의 값을 1로 설정했습니다. 즉, 적어도 하나 이상의 문서에서 출현한 단어는 모두 사용해서 문서를 벡터로 표현하겠다는 것을 의미합니다.

from sklearn.feature\_extraction.text import CountVectorizer

tf\_vectorizer = CountVectorizer(min\_df=1, max\_df=0.8, ngram\_range=(1,1))

DTM\_tf = tf\_vectorizer.fit\_transform(documents\_filtered)

그리고 이를 numpy의 어레이 형태로 변환합니다.

import numpy as np

DTM\_TF = np.array(DTM\_tf.todense())

TF 기반의 DTM 정보를 저장하고 있는 DTM\_TF 변수의 형태 (shape)을 보도록 하겠습니다.

DTM\_TF.shape

(4, 285)

이는 우리가 구한 DTM이 4x285 형태의 행렬이라는 것을 의미합니다. 즉, 행의 수가 4이고 열의 수가 285인 행렬입니다. DTM의 행의 수는 문서의 수를 의미하고, 열의 수는 문서를 벡터로 표현할 때 사용된 단어의 수를 의미합니다. 여기서는 문서를 벡터로 표현할 때 사용된 단어의 수가 285개라는 것을 의미하며, 이는 각 벡터가 285개의 원소를 갖는 벡터 (즉, 285차원 벡터)라는 것을 뜻합니다.

이러한 벡터를 이용해서 첫 번째 문서와 나머지 세 개의 문서들 간의 유클리디안 거리를 계산해 보겠습니다.

print(np.linalg.norm(DTM\_TF[1]-DTM\_TF[0])) # 첫 번째 문서와 두 번째 문서 간 유클리디안 거리

print(np.linalg.norm(DTM\_TF[2]-DTM\_TF[0])) # 첫 번째 문서와 세 번째 문서 간 유클리디안 거리

print(np.linalg.norm(DTM\_TF[3]-DTM\_TF[0])) # 첫 번째 문서와 네 번째 문서 간 유클리디안 거리

23.895606290697042

24.919871588754223

25.826343140289914

차이는 크게 나지 않지만, 첫 번째 문서와 두 번째의 문서의 유사도가 제일 큰 것으로 나왔습니다 (앞 부분에서 이야기한 것 처럼 두 문서는 모두 애플 아이폰에 대해서 다루고 있습니다).

이번에는 코사인 유사도를 계산해 보도록 하겠습니다.

print(np.dot(DTM\_TF[0],DTM\_TF[1])/(np.linalg.norm(DTM\_TF[0])\*np.linalg.norm(DTM\_TF[1])))

print(np.dot(DTM\_TF[0],DTM\_TF[2])/(np.linalg.norm(DTM\_TF[0])\*np.linalg.norm(DTM\_TF[2])))

print(np.dot(DTM\_TF[0],DTM\_TF[3])/(np.linalg.norm(DTM\_TF[0])\*np.linalg.norm(DTM\_TF[3])))

0.5959464513198515

0.010149858103156184

0.021112128709387182

코사인 유사도의 결과도 첫 번째 문서와 두 번째 문서 간의 유사도가 제일 크게 나왔습니다. 다만, 유클리디안 거리 결과에 비해서 그 차이가 상대적으로 더 크게 나왔습니다.

이번에는 TfidfVectorizer 클래스를 이용해서 벡터화 작업을 수행해 보도록 하겠습니다.

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf\_vectorizer = TfidfVectorizer(min\_df=1, max\_df=0.8, ngram\_range=(1,1))

DTM\_tfidf = tfidf\_vectorizer.fit\_transform(documents\_filtered)

DTM\_TFIDF = np.array(DTM\_tfidf.todense())

TF-IDF 기반의 DTM 정보가 DTM\_TFIDF 변수에 저장되어 있습니다. 벡터화할 때 동일한 조건을 사용했기 때문에 DTM\_TFIDF의 형태(shape)는 앞에서의 경우와 동일합니다. 즉, 4x285입니다.

DTM\_TFIDF.shape

(4, 285)

이번에도 역시나 마찬가지로 유클리디안 거리와 코사인 유사도를 이용해서 첫 번째 문서와 나머지 문서들 간의 유사도를 계산해 보겠습니다. 먼저 유클리디안 거리를 이용한 결과입니다.

print(np.linalg.norm(DTM\_TFIDF[1]-DTM\_TFIDF[0]))

print(np.linalg.norm(DTM\_TFIDF[2]-DTM\_TFIDF[0]))

print(np.linalg.norm(DTM\_TFIDF[3]-DTM\_TFIDF[0]))

0.9538556951497862

1.409374356422095

1.4028109418615884

첫 번째 문서와 두 번째 문서 간의 유클리디안 거리가 TF 정보를 이용하는 경우 보다 상대적으로 다른 값들에 비해서 작게 나온 것을 확인할 수 있습니다.

이번에는 코사인 유사도 결과를 확인해 보겠습니다.

print(np.dot(DTM\_TFIDF[0],DTM\_TFIDF[1])/(np.linalg.norm(DTM\_TFIDF[0])\*np.linalg.norm(DTM\_TFIDF[1])))

print(np.dot(DTM\_TFIDF[0],DTM\_TFIDF[2])/(np.linalg.norm(DTM\_TFIDF[0])\*np.linalg.norm(DTM\_TFIDF[2])))

print(np.dot(DTM\_TFIDF[0],DTM\_TFIDF[3])/(np.linalg.norm(DTM\_TFIDF[0])\*np.linalg.norm(DTM\_TFIDF[3])))

0.5450796564151593

0.006831961729902576

0.01606073069670202

이번에도 그 차이가 TF 정보를 이용했을 때 보다 상대적으로 더 크게 나왔습니다. 이러한 결과는 TF-IDF 정보를 기반으로한 벡터가 TF 기반 벡터 보다 문서의 고유한 특성을 더 잘 나타낸다는 것을 의미한다고 볼 수 있습니다.

# 군집 분석 (clustering analysis)

자, 지금까지 문서를 벡터로 바꾸고 벡터간의 유사도를 계산하여 문서의 유사도를 파악하는 방법에 대해서 알아보았습니다. 그렇다면 우리는 문서 간의 유사도를 가지고 무엇을 할 수 있을까요? 여러가지를 할 수 있지만, 대표적으로 할 수 있는 것이 유사한 문서들을 같은 그룹으로 묶어주는 작업입니다. 이러한 작업을 군집화 (clustering)이라고 합니다. 군집화는 유사도가 더 큰 문서들을 같은 군집으로 묶는 작업을 의미합니다. 그리고 문서들의 유사도를 계산하기 위해서 각 문서를 벡터로 변환한 후 유클리디안 거리나 코사인 유사도를 사용해서 벡터 간의 유사도를 계산합니다. 우리가 이미 모두 배운 것이죠. 이번 장에서는 벡터의 유사도 개념을 기반으로 군집화를 수행하는 기계학습 알고리즘들에 대해서 살펴보도록 하겠습니다.

군집분석 (clustering analysis)은 여러개의 문서들을 서로 유사한 문서들끼리 같은 그룹 (cluster)로 묶어주는 것입니다. 문서 간의 유사한 정도는 사용된 단어가 얼마나 비슷하냐에 따라 달라집니다. 즉, 비슷한 단어가 비슷한 정도로 사용된 문서 간의 유사도가 높은 것입니다. 군집 분석도 기계학습을 통해 이루어지는데 비지도학습 방법을 이용합니다. 즉, 별도의 종속변수 혹은 레이블 정보가 없는 여러개의 문서로 구성된 텍스트 데이터에 대해서 문서에 사용된 단어를 분석하여 비슷한 단어들이 사용된 문서끼리 군집으로 묶어주는 분석입니다. 예를 들어, 여러개의 신문기사로 구성이 된 텍스트 데이터가 있는 경우에, 해당 신문기사들이 ‘정치’, ‘경제’, ‘스포츠’ 섹션에 속하는 것들이고 어느 섹션에 속한 기사인지에 따라 사용되는 단어에 차이가 있다고 가정을 한다면, 기사들에 대한 ‘정치’, ‘경제’, ‘스포츠’ 등의 섹션 정보 (즉, 종속변수 또는 레이블 정보)가 없더라도, 군집 분석을 사용해서 비슷한 문서끼리 묶어줌으로써 각 기사가 속한 섹션의 정보를 발견할 수 있습니다. 뿐만 아니라, 하나의 섹션 (예, 정치 섹션)에 속하는 기사들이라고 하더라도 세부적으로 다루고 있는 주제에 따라서 사용되는 단어들이 다를 수 있습니다. 이런 경우, 군집 분석을 통해서 같은 정치 섹션에 포함된 기사들이라고 하더라도 세부적으로 어떠한 주제들을 다루고 있는지, 그리고 각 주제를 다루고 있는 기사들에는 무엇이 있는지를 확인할 수 있는 것입니다.

군집화 분석 순서는 다음과 같습니다.

① 텍스트 데이터 준비

분석에 사용될 텍스트 데이터를 준비해야 합니다. 온라인에서 수집하는 경우라면 앞에서 설명된 web scraping 방법을 사용할 수 있습니다. Web scraping 방법을 통해서 특정한 기준을 만족하는 텍스트 데이터를 1차적으로 전부 수집하고, 그 후에 세부적인 과정을 거쳐서 최종 분석에 필요한 데이터를 선별하는 작업이 필요합니다. 군집 분석은 비지도학습이기 때문에 학습데이터를 따로 준비할 필요가 없습니다.

② 텍스트 전처리와 문서의 벡터화

온라인에서 수집된 텍스트 데이터를 군집 분석에 사용될 수 있도록 전처리 과정을 거쳐야 합니다. 전처리 과정을 통해서 불용어가 제거된 특정 품사 (보통은 명사, 동사, 형용사)를 갖는 단어들만을 선택하여 최종 분석에 사용하게 됩니다. 그 다음, 선택된 단어들을 가지고 각 문서를 벡터로 표현해야 합니다. 즉, 전체 문서 데이터를 DTM 로 변환을 해야하는데, Frequency-based DTM이나 TF-IDF DTM 형태로 변환할 수 있습니다.

③ 기계학습 알고리즘을 통한 군집화

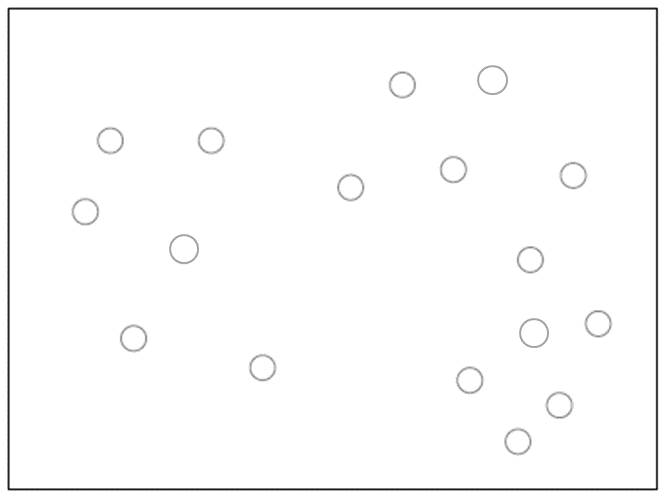
다양한 기계학습 알고리즘이 텍스트 군집화에 사용될 수 있지만, 그 중에서 많이 사용되는 것에는 K-means와 위계적 군집화 (Hierarchical clustering) 방법이 입니다. 두 알고리즘 모두 벡터 형태로 표현이 된 각 문서간의 거리를 계산하여 가까운 거리에 위치하는 문서끼리 묶어주는 역할을 합니다.

## K-Means 알고리즘

벡터 간의 유사도에 대한 기본적인 이해를 토대로 지금부터는 군집화 분석에 사용되는 기계학습 알고리즘들을 살펴보도록 하겠습니다. 첫번째로 살펴보고자 하는 알고리즘은 K-Means 알고리즘입니다.

#### K-Means 작동 원리

K-Means 알고리즘에서는 문서 간의 거리를 계산할 때 유클리디안 방법이 주로 사용됩니다. K-Means를 통해서 군집화가 되는 방식은 비교적 간단합니다. K-Means를 설명하기 위해 아래와 같이 여러 개의 문서들이 2차원 공간 상에 존재한다라고 가정하겠습니다 (실제로는 문서가 존재하는 공간의 차원수는 문서를 벡터로 만들 때 사용한 단어의 수와 같습니다. 하지만, 여기서는 설명을 위해서 각 문서가 2차원 공간의 벡터로 표현된다고 가정합니다).



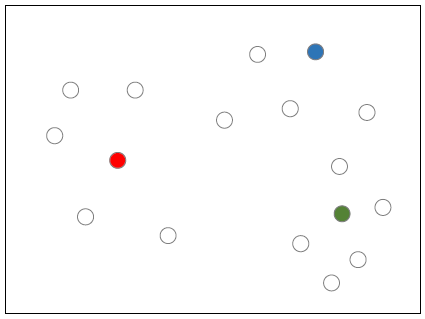
위와 같은 문서 벡터들에 대해서 K-Means 알고리즘은 아래와 같은 순서로 작동합니다.

Step 1) 찾고자 하는 군집의 수 (=K)를 임의로 정하기 (군집의 수를 정하는데 사용되는 방법에 대해서는 다음 서브섹션에서 설명합니다)

K-Means 알고리즘을 사용하기 위해서는 일단 먼저 사용자가 찾고자 하는 군집의 수 (=K)를 지정해야 합니다. 그러면 K-Means알고리즘은 (사용자가 정한) K개의 군집을 문서 간의 유사성에 따라 찾아 줍니다. 여기서는 설명을 위해 찾고자 하는 군집의 수를 3이라고 정했다고 가정합니다 (즉, K=3).

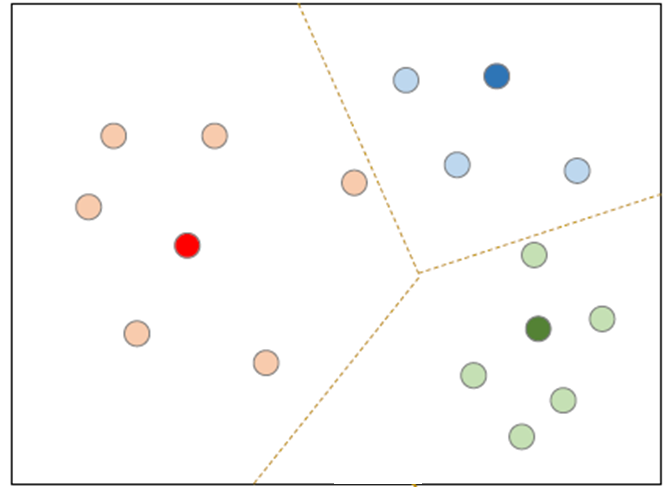
Step 2) 각 군집의 중심 벡터 정하기

찾고자 하는 군집의 수가 3이라고 정해지면, K-Means 알고리즘은 처음에 임의로 3개의 문서를 선택합니다. 그 3개의 문서가 각 군집의 일시적인 중심이 되는 것입니다 (아래 그림 참고).



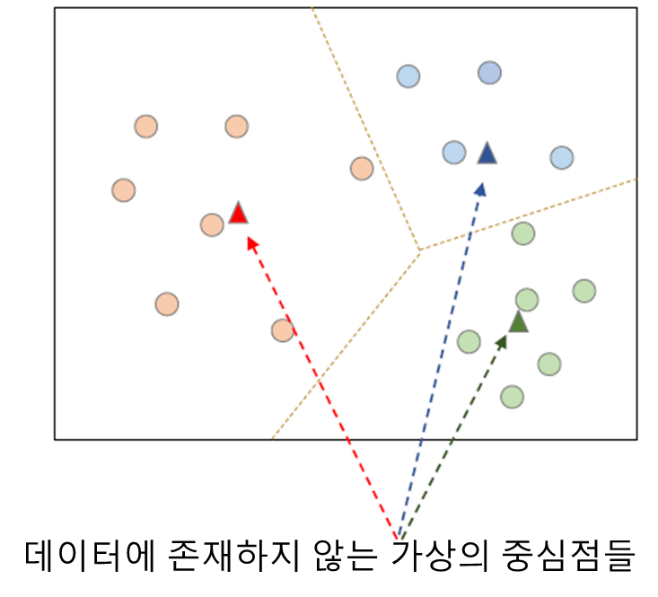
Step 3) 각 문서가 속하는 군집 정하기

그 다음 문서 데이터에 존재하는 각 문서 벡터에 대해서 3개의 중심 중에서 제일 가까이 위치하는 중심을 찾습니다. 각 문서 (혹은 문서 벡터)가 속하는 군집은 가장 가까이 위치해 있는 중심이 속한 군집이 되는 것입니다 (아래 그림 참고).



Step 4) 각 군집의 새로운 중심 정하기

임시적인 군집이 형성되었습니다. 이번에는 각 군집에 속한 각 문서 벡터들의 평균 값으로 그 군집의 중심점을 새롭게 선정합니다. 이렇게 되면 실제 문서가 각 그룹의 중심이 되는 것이 아니라 가상의 중심점이 생기게 되는 것이죠 (아래 그림 참고).



Step 5) 각 문서가 속하는 군집 정하기

새롭게 정해진 중심에 대해서 Step3의 방법과 동일한 방법을 사용하여 다시 한번 각 문서가 속하는 군집을 결정하게 됩니다. 이전 단계의 중심에서 새로운 중심으로 이동하였기 때문에, 일부 문서들이 속한 군집이 달라질 수 있습니다.

Step 6) 앞의 단계에서 군집이 달라진 문서들이 있으면 Step4 ~ 5) 과정을 반복합니다. 더 이상 군집이 바뀌는 문서가 없을 때까지 반복하게 됩니다 (혹은 사용자가 정한 횟수 만큼 반복하게 됩니다).

#### 군집의 수 정하기

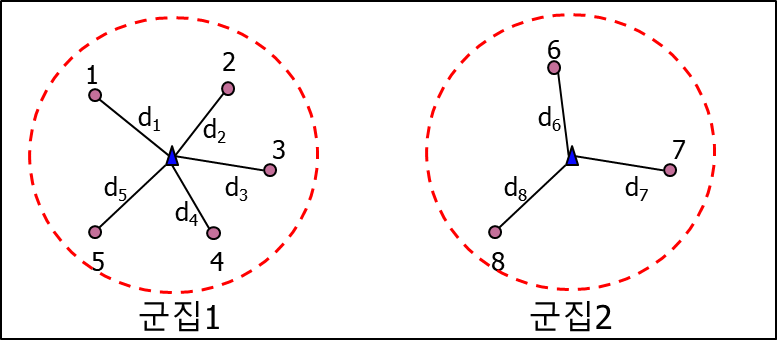
군집의 수를 정하기 위해서 사용되는 방법에 크게 두 가지가 있습니다. 하나가 엘보우 방법이고, 다른 하나가 Silhouette 값을 사용하는 방법입니다.

**① 엘보우 방법 (Elbow method)**

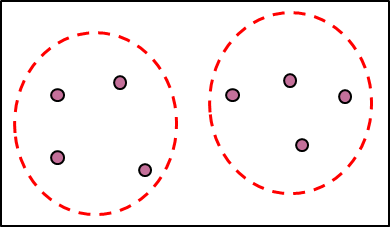
엘보우 방법은 Score 값을 사용하는 방법입니다. Score 값은 다음과 같이 정의됩니다.

는 데이터 포인트 i와 데이터 포인트 i가 속한 군집의 중심 간의 거리입니다.

만약 전체 데이터 포인트의 수가 7개 이고, 7개의 데이터 포인트가 서로 다른 2개의 군집에 속해 있다고 가정하겠습니다 (아래 그림 참고). 이 같은 경우 각 군집의 중심은 아래 그림에서와 같이 파란색 삼각형으로 표현될 수 있습니다. 아래 그림에 대한 score 값은 (즉, 으로 계산됩니다.

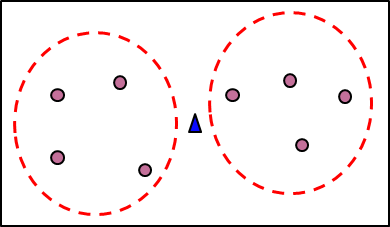


보다 직관적인 설명을 위해서 score 값에서 –를 제외한 부분만 사용해서 엘보우 방법을 설명하겠습니다. 군집의 수에 따라서 값이 어떻게 달라지는지를 살펴보겠습니다. 설명을 위해서 8개의 데이터 포인트로 구성된 데이터셋이 존재한다고 가정합니다. 그리고 True 군집의 수는 2개라고 가정합니다 (아래 그림 참고).

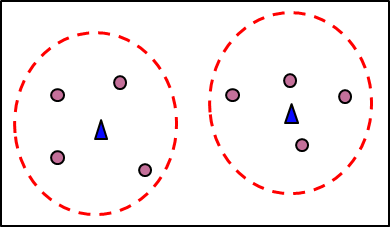


우리가 찾아야 하는 군집의 수는 2가 되는 것입니다. 이를 위해서 군집의 수를 1부터 시작해서 하나씩 증가시키면서 값이 어떻게 달라지는지를 살펴보겠습니다.

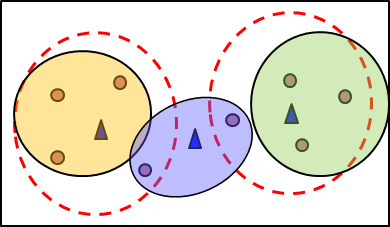
군집의 수 = 1 인 경우에 해당 군집의 중심은 아래 그림과 같이 표기될 수 있습니다.



이번에는 군집의 수를 하나 증가시켜 보게습니다. 그러면 각 군집의 중심은 아래와 같이 표기될 것입니다.

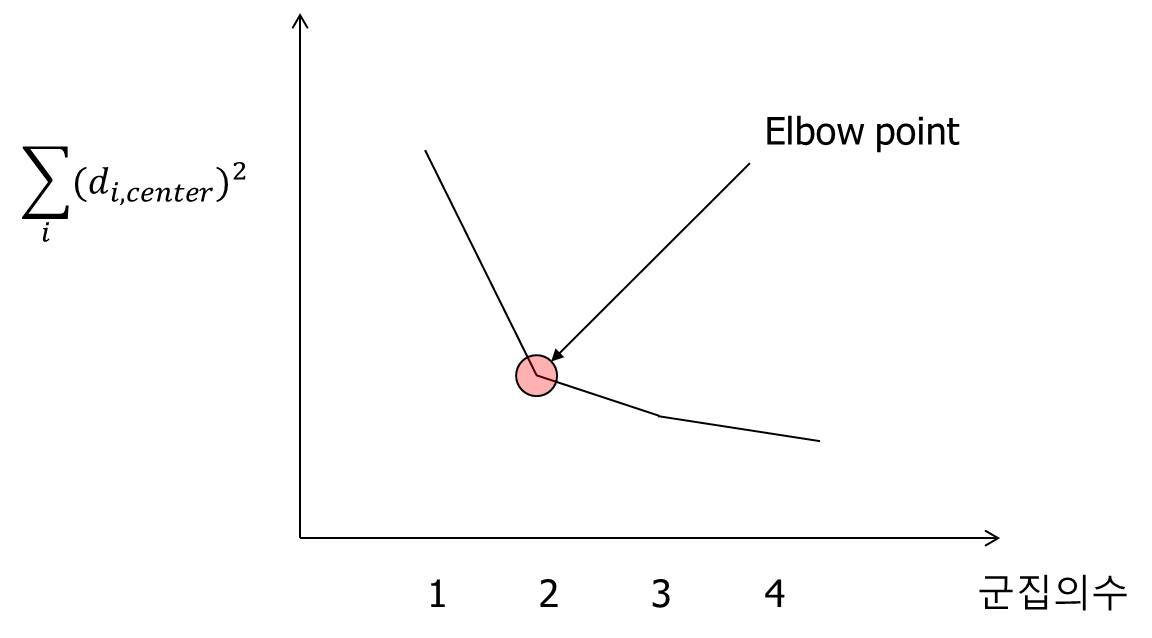


군집의 수가 1에서 2로 증가함에 따라 값이 크게 줄어드는 것을 알 수 있습니다. 왜냐하면 각 데이터 포인트와 해당 데이터 포인트가 속한 군집의 중심간의 거리가 크게 줄기 때문입니다. 여기에서 군집의 수를 하나 증가시켜 보겠습니다. 그러면 아래와 같이 군집의 중심이 표기될 수 있습니다.



군집의 수가 2에서 3으로 증가하는 경우에는 중심과 데이터 포인트 간의 거리가 크게 줄어들지 않습니다. 즉, 값이 조금만 줄어드는 것입니다. 군집의 수를 하나 더 늘려도 마찬가지로 는 조금 밖에 줄어들지 않을 것입니다.

이제 각 군집 수에 따라서 값을 그래프로 표현해 보겠습니다.



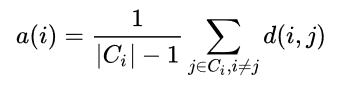
위의 그림에서 보는 것 처럼 군집의 수가 1에서 2로 증가하는 경우에 이 크게 줄어들고, 그 다음부터는 군집의 수가 증가해도 가 줄어드는 정도가 줄어듭니다. 그리고 여기에서 여러분들이 기억해야 하는 것은 우리가 찾고자 하는 실제 군집의 수는 2라는 것입니다. 즉, 최적의 군집 수 이전에는 가 크게 줄어들고, 그 이후에는 조그만 줄어든다는 것입니다. 우리는 그래프가 급격하게 꺽이는 점 (위 그림에서의 Elbow point에 해당 하는 점)에 해당 하는 군집의 수를 최적의 군집의 수로 선택하게 되는데 그 모양이 팔꿈치와 비슷해서 이러한 방법을 elbow 방법이라고 합니다.

**② Silhouette score 사용하기**

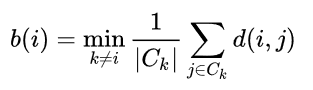
실루엣 값은 하나의 데이터 포인트 (즉, 하나의 문서)에 대해서 정의됩니다. 특정 데이터 포인트에 대해서 해당 데이터 포인트가 속한 군집의 다른 데이터 포인트들과의 평균 거리와 해당 데이터 포인트가 속하지 않은 다른 군집에 속한 데이터 포인트들 간의 평균 거리와의 차이를 보는 것입니다. 값이 클수록 같은 군집에 속한 데이터 포인트들하고의 거리는 가깝고 다른 군집에 속한 데이터 포인트들하고의 거리는 거리가 멀다는 것을 의미합니다. 즉, 값이 클수록 군집화가 잘 되었다라는 것을 의미합니다. 클러스터 i (라고 표현)에 속한 데이터 포인트 i의 실루엣 값은 다음과 같이 계산됩니다.

 또는 

여기에서 는 클러스터 i (라고 표현)에 속한 데이터 포인트의 수가 됩니다. 와 는 아래와 같이 정의됩니다.

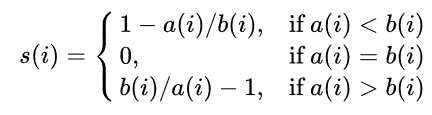


는 데이터 포인트 와 같은 군집에 속한 다른 데이터 포인트들 간의 평균 거리입니다 (시그마아래 부분에 있는 가 의미하는 것은 군집 에 속한 데이터 포인트들 중에서 데이터 포인트 가 아닌 다른 모든 데이터 포인트들을 의미합니다). 의 값이 작을 수록 같은 군집에 속한 다른 포인트들과의 유사도가 높다는 것을 의미합니다.



는 데이터 포인트 i가 속하지 않은 군집들 중에서 평균 거리가 가장 가까운 군집에 속해 있는 데이터 포인트들과의 (데이터 포인트 i와의) 평균 거리를 의미합니다.

위 실루엣 값의 정의는 아래와 같이 다시 표현될 수 있습니다.



이는 을 의미합니다. 1에 가까울수록 군집화가 잘 되었다는 것을 의미합니다.

군집 의 실루엣 값은 에 속한 모든 데이터 포인트들이 갖는 실루엣 값의 평균으로 계산됩니다.

Note!

엘보우 방법이나 실루엣 값을 사용하는 방법이 항상 군집의 수를 정확하게 파악해 주는 것은 아닙니다. 처음부터 한번에 최적의 군집의 수를 찾기 보다는, 순차적인 방법으로 최적의 군집의 수를 찾는 것도 생각해 볼 수 있습니다. 예를 들어, 찾고자 하는 군집의 수를 되도록 크게 한 다음 (예, K = 100), 기계학습 알고리즘을 통해서 선별이 된 K개의 군집에 대해서 추가적인 군집화를 진행하는 것이입니다. 각 군집에 속한 문서들에 대해서 추가적인 유사도 검사를 해서 더 유사한 문서끼리 묶어주는 작업을 추가적으로 진행할 수 있습니다.

#### 파이썬 코딩하기[[35]](#footnote-36)

일단 아래와 같이 텍스트 데이터를 불러옵니다. 여기서는 서로 다른 5개의 주제를 다루고 있는 15개의 기사들로 구성된 데이터를 사용합니다. 각각의 기사는 하나의 주제만을 다룹니다. 예를 들어, 첫번째 3개의 기사는 첫번째 주제에 대해서 다루고 있습니다.

import numpy as np

from os import listdir

from os.path import isfile, join

mypath = './example\_En\_docs/'

onlyfiles = [f for f in listdir(mypath) if isfile(join(mypath, f))]

total\_docs = []

for file in onlyfiles:

file\_path = mypath+file

with open(file\_path, 'r', encoding='utf8') as f:

content = f.read()

total\_docs.append(content)

위의 코드를 실행하면, total\_docs 변수에는 15개의 원소가 존재하게 되고, 각 원소는 각 기사의 원본 내용을 문자열 형태로 가지고 있습니다. 군집화 분석 알고리즘을 적용하기 이전에 일단 먼저 전처리 작업을 해야 합니다. 전처리 작업을 통해서 불용어가 제거된 특정한 품사의 단어들만을 선택해서 각 문서를 표현하게 되는데, 군집화 분석에서는 보통 명사의 단어들을 선택합니다. 그 이유는 많은 경우에 우리는 군집화 분석을 통해서 유사한 주제를 다루고 있는 문서들을 그룹핑하고자 하기 때문입니다. 영어 텍스트 전처리 작업을 위해서 nltk를 사용합니다. 전처리에 필요한 함수가 preprocessing.py에 저장되어 있습니다. 해당 함수를 사용하기 위해서 아래와 같이 preprocessing.py 파일을 임포트 합니다.

import preprocessing # import a python file for English text preprocessing

해당 파일에 저장되어 있는 En\_preprocessing() 함수를 사용해서 전처리를 하는데, 해당 함수는 두개의 인자를 입력받습니다. 하나는 원본 문자열 데이터이고, 다른 하나는 사용자 정의 불요어 함수입니다. 다음과 같이 사용할 수 있습니다.

stopwords = ['be', 'today', 'yesterday', 'new', 'york', 'time'] # 불용어 사전 생성하기

docs\_nouns = [preprocessing.En\_preprocessing(doc, stopwords) for doc in total\_docs]

docs\_nouns 는 리스트 변수로 각 원소가 각 문서의 명사 단어들을 포함하고 있는 리스트가 됩니다. 아래와 같이 첫번째 원소를 확인해 보겠습니다.

In: print(docs\_nouns[0])

Out: ['aleman', 'world', 'video', 'game', 'front', 'gunshot', 'restroom', 'cover', 'mass', 'shooting', 'florida', 'tournament', 'player', 'football', 'video', 'game', 'madden', 'jacksonville', 'winner', 'level', 'tournament', 'vega', 'october', 'cash', 'prize', 'participant', 'jacksonville', 'marketplace', 'store', 'bar', 'restaurant', 'st', 'john', 'river', 'david', 'gamer', 'baltimore', 'maryland', 'jacksonville', 'tournament', 'game', 'bar', 'back', 'pizza', 'restaurant', 'sunday', 'gun', 'venue', 'fire', 'people', 'gun', 'mike', 'williams', 'police', 'motive', 'katz', 'handgun', 'sheriff', 'people', 'wound', 'people', 'area', 'williams', 'victim', 'condition', 'hospital', 'sheriff', 'horror', 'stream', 'event', 'game', 'shot', 'people', 'person', 'f', 'whatd', 'call', 'william … ] # 일부 결과 생략

즉, 첫번째 문서 (신문기사)를 구성하고 있는 명사 단어들로 구성된 리스트 데이터인 것을 확인할 수 있습니다. 이렇게 준비된 데이터를 sklearn에서 제공되는 CountVectorizer 또는 TfidfVectorizer 클래스를 이용해서 DTM (document-term matrix)로 변환합니다. docs\_nouns의 경우 각 문서가 리스트의 형태로 되어 있는데, CountVectorizer나 TfidfVectorizer를 사용하기 위해서는 각 문서를 문자열의 형태로 변환해야 합니다. 아래의 코드를 실행합니다.

documents\_filtered = []

for doc in docs\_nouns:

document\_filtered = ' '.join(doc)

documents\_filtered.append(document\_filtered)

# 불필요한 단어들을 제거하고 난후 DTM로 변환하기 위해 다시 list of strings의 형태로 변환

DTM을 만드는 방법은 앞에서 설명했으니 여기서는 코드만 살펴보도록 하겠습니다. 여기서는 TF-IDF DTM을 생성합니다.

from sklearn.feature\_extraction.text import TfidfVectorizer # tf-idf based DTM

vectorizer = TfidfVectorizer()

DTM\_tfidf = vectorizer.fit\_transform(documents\_filtered) # 문서들을 TFIDF matrix로 변환

그런 다음 다음과 같이 numpy의 array 형태로 변환합니다.

DTM\_TFIDF = np.array(DTM\_tfidf.todense())

DTM\_TFIDF는 TF-IDF 값을 가지고 있는 DTM이 됩니다. 즉, 각 행이 각 문서의 벡터가 되는 것입니다. 예를 들어, DTM\_TFIDF[0]은 첫번째 문서 (즉, 첫번째 신문기사)에 대한 벡터가 됩니다. 우리는 이러한 각 문서에 대한 벡터를 이용해서 벡터간의 유사도를 계산할 수 있습니다. 유클리디안 거리는 아래와 같이 계산합니다. 아래 코드에서는 첫번째 기사와 두번째 기사, 4번째 기사, 그리고 6번째 기사 간의 유클리디안 거리를 계산하고 있습니다. 첫번째 기사와 두번째 기사가 동일한 주제에 대해서 다루고 있기 때문에 상대적으로 유클리디안 거리가 더 짧게 나온 것을 확인할 수 있습니다.

d1d2\_tfidf = np.linalg.norm(DTM\_TFIDF[0]-DTM\_TFIDF[1])

d1d4\_tfidf = np.linalg.norm(DTM\_TFIDF[0]-DTM\_TFIDF[3])

d1d6\_tfidf = np.linalg.norm(DTM\_TFIDF[0]-DTM\_TFIDF[5])

print(d1d2\_tfidf, d1d4\_tfidf, d1d6\_tfidf)

Out: 1.0862427819879072 1.3761591218900362 1.3995826682418473

이번에는 코사인 유사도를 계산해 보겠습니다.

d1d2\_cos\_tfidf = np.dot(DTM\_TFIDF[0],DTM\_TFIDF[1])/(np.linalg.norm(DTM\_TFIDF[0])\*np.linalg.norm(DTM\_TFIDF[1]))

d1d4\_cos\_tfidf = np.dot(DTM\_TFIDF[0],DTM\_TFIDF[3])/(np.linalg.norm(DTM\_TFIDF[0])\*np.linalg.norm(DTM\_TFIDF[3]))

d1d6\_cos\_tfidf = np.dot(DTM\_TFIDF[0],DTM\_TFIDF[5])/(np.linalg.norm(DTM\_TFIDF[0])\*np.linalg.norm(DTM\_TFIDF[5]))

print(d1d2\_cos\_tfidf, d1d4\_cos\_tfidf, d1d6\_cos\_tfidf)

Out: 0.41003830928958646 0.053093035619423086 0.020584177378516227

이번에도 유클리디안 거리의 결과와 유사한 결과가 나온 것을 알 수 있습니다. 다만, 상대적인 차이가 더 명확하게 나왔습니다.

그 다음에 해야 하는 것은 준비된 데이터 (즉, DTM\_TFIDF)에 KMeans 알고리즘을 적용하는 것입니다. 이를 위해서 sklearn에서 제공되는 KMeans 클래스를 사용합니다. 아래와 같이 임포트를 하고 생성자 함수를 이용해서 해당 클래스의 객체를 만듭니다. 이때 생성자 함수의 파라미터인 n\_cluster의 값을 지정해야 합니다. 이는 우리가 찾고자 하는 군집의 수가 됩니다. 우리가 다루고 있는 데이터에 존재하는 신문기사들의 경우 서로 다른 5개의 주제를 다루고 있으므로 우리가 찾고자 하는 군집의 수는 5가 됩니다 (하지만 많은 군집화 분석에서는 정답의 군집 수를 사전에 알지 못합니다. 그러한 경우에는 elbow 방법이나 silloette score를 사용해야 합니다. 이와 관련된 코드는 잠시 후에 설명하겠습니다). 객체를 생성한 후 fit\_predict() 함수를 이용해서 군집화 분석을 합니다.

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=5)

clusters = kmeans.fit\_predict(DTM\_TFIDF) # KMeans 알고리즘으로 학습

clusters에는 군집화 분석의 결과가 저장되어 있습니다.

In: print(clusters)

Out: [2 2 2 4 4 3 3 3 1 1 1 1 0 0 0]

위의 결과에서 각 숫자가 의미하는 것은 군집의 ID가 됩니다.[[36]](#footnote-37) 우리가 찾은 전체 군집의 수가 5이므로 군집의 ID는 0 부터 4까지의 값을 갖습니다. 첫번째 숫자인 2는 첫번째 문서가 속한 군집의 ID가 됩니다. 즉, 첫번째 문서는 군집 2에 속해 있습니다. 그리고, 두번째, 세번째 문서들도 ID가 2인 군집 (즉, 군집 2)에 속한 것을 알 수 있습니다. 어떠한 문서가 어떠한 군집에 속한지는 별로 중요한지 않습니다. 그 보다 중요한 것이 어떠한 문서들이 동일한 군집에 속해 있느냐 입니다. 동일한 군집에 속한 신문기사들이 동일한 주제 혹은 유사한 주제에 대해서 다루고 있을 가능성이 큰 것입니다.

각 군집의 중심을 아래와 같이 확인할 수 있습니다. 군집의 중심도 공간상의 하나의 점 즉, 벡터가 됩니다 (해당 벡터의 크기는 각 문서 벡터의 크기와 동일합니다. 본 예에서의 각 문서 벡터의 크기는 1165입니다. 즉, 각 문서가 1165의 단어들을 이용해서 표현되었습니다).

In: len(kmeans.cluster\_centers\_[0])

Out: 1165

In: kmeans.cluster\_centers\_ # 각 클러스터의 중심 구하기

Out: array([[0. , 0. , 0. , ..., 0. , 0. ,

0. ],

[0.01570475, 0.01363693, 0.01570475, ..., 0. , 0. ,

0. ],

[0. , 0. , 0. , ..., 0.01789861, 0.02682787,

0.02682787],

[0. , 0.01388332, 0. , ..., 0. , 0. ,

0. ],

[0. , 0. , 0. , ..., 0. , 0. ,

0. ]])

각 군집의 중심을 이용해서 각 군집에 속한 문서 벡터가 군집의 중심과 얼마나 떨어져 있는지 확인할 수 있습니다.

for i, k in enumerate(clusters):

dis = np.linalg.norm(DTM\_TFIDF[i]-kmeans.cluster\_centers\_[k])

print('Doc: {0}, Cluster: {1}, Distance to the center: {2}'.format(i, k, dis))

이번에는 엘보우 방법과 실루엣 값을 이용해서 최적의 군집 수를 찾아보도록 하겠습니다.

엘보우 방법

앞에서 설명한 것 처럼 엘보우 방법은 각 군집의 수에 대해서 score값을 계산하고, 그래프로 그려서 최적의 군집 수를 찾는 방법입니다. 각 군집 수에 대한 score 값은 KMeans에서 제공하는 score() 함수를 사용해서 계산합니다. 해당 함수는 인자로 DTM을 입력받습니다.

In: kmeans.score(DTM\_TFIDF)

Out: -5.362977736502234

위의 kmeans 객체는 군집의 수를 5로 해서 생성된 객체이기 때문에 위의 score() 함수가 출력하는 값은 군집의 수 = 5인 경우에 대한 값이 됩니다.

찾고자 하는 군집의 수를 1부터 10까지 변경해 가면서 score값을 구하고 그 결과를 시각화해 보겠습니다.

number\_clusters = range(1, 11)

kmeans\_list = []

for i in number\_clusters:

kmeans\_list.append(KMeans(n\_clusters=i))

scores = []

for i in range(len(kmeans\_list)):

scores.append(kmeans\_list[i].fit(DTM\_TFIDF).score(DTM\_TFIDF))

# score = - sum of sq

# score indicates the variance explained by the clusters

pl.plot(number\_clusters,scores)

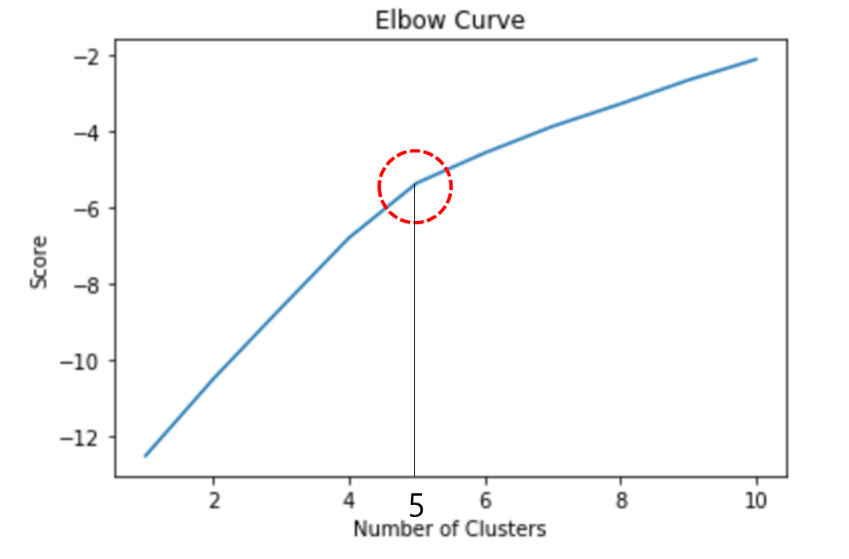
pl.xlabel('Number of Clusters')

pl.ylabel('Score')

pl.title('Elbow Curve')

pl.show()

시각화의 결과는 아래와 같습니다. 그래프가 급하게 꺽이는 지점에서의 군집의 수가 최적의 군집의 수일 확률이 높다라고 얘기했습니다. 아래 그림에서는 군집의 수 = 5 인 지점에서 그래프가 가장 심하게 꺽인 것을 확인할 수 있습니다. 즉, 아래 그림을 토대로 한다면 최적 군집의 수는 5가 되는 것입니다.



실루엣 스코어

이번에는 실루엣 스코어 값을 확인해 보겠습니다. 이를 위해서 sklearn.metrics에서 제공하는 silhouette\_score() 함수를 사용합니다. 여기서도 군집의 수를 1부터 10까지 변경해 가면서 실루엣 스코어를 계산해 보겠습니다.

from sklearn.metrics import silhouette\_score

for k in range(2,11):

kmeans = KMeans(n\_clusters=k, max\_iter=10)

cluster = kmeans.fit\_predict(DTM\_TFIDF)

print(k, silhouette\_score(DTM\_TFIDF, cluster))

결과는 아래와 같습니다. 군집의 수 = 5인 경우에 실루엣 스코어가 가장 큰 것을 확인할 수 있습니다. 이 결과 또한 최적 군집의 수는 5라는 것을 의미합니다.

2 0.09491019881878285

3 0.15573802952332308

4 0.20372920795120109

5 0.24272695180620912

6 0.22109589941439053

7 0.19477621495628059

8 0.19140320778201333

9 0.18304440127896748

10 0.14399663475557473

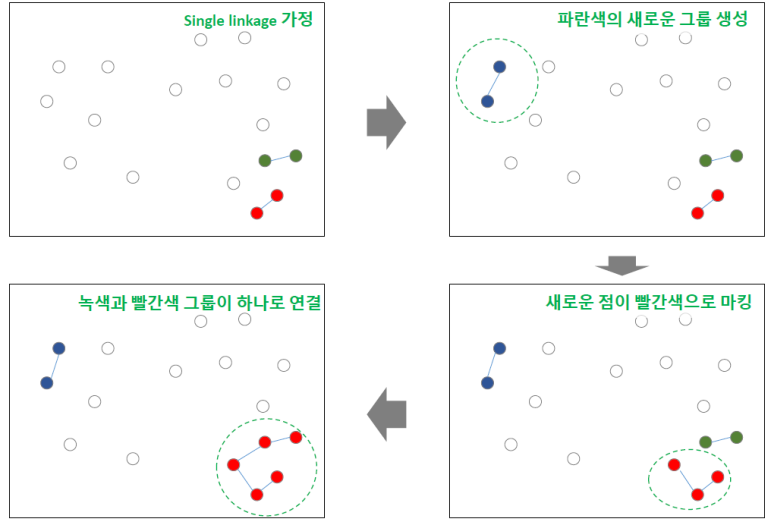
## 위계적 군집 분석 (hierarchical clustering analysis)

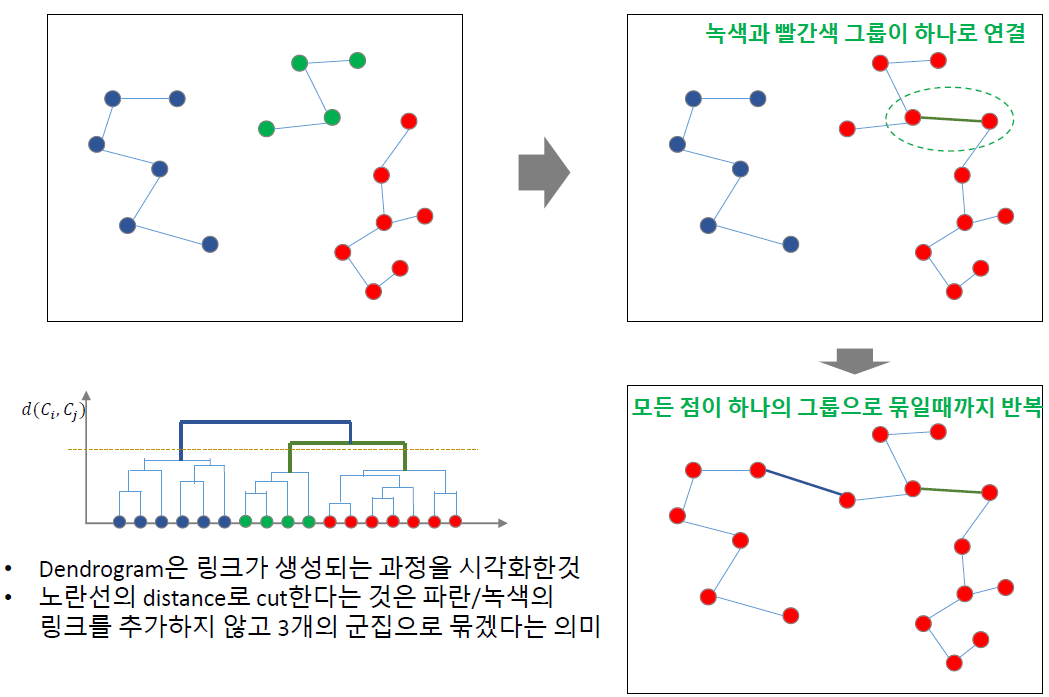
K-Means와 더불어서 가장 일반적으로 많이 사용되는 군집 분석 방법이 위계적 군집 분석 방법입니다. 위계적 군집 분석에서는 각 데이터 포인트를 하나의 군집이라고 생각하고 시작을 합니다. 즉, 처음에는 하나의 군집안에 하나의 데이터 포인트만이 속해 있다고 생각할 수 있습니다.

#### 위계적 군집 분석 설명

위계적 군집 분석 방법에는 크게 2가지 종류가 있습니다. 하나는 Agglomerative 방법입니다. 이는 "bottom-up" 이라고 볼 수 있는데, 처음에 각 데이터 포인트가 하나의 군집이라고 가정하고 군집들을 순차적으로 묶어주면서 진행되는 방법입니다. 다른 하나는 Divisive 방법입니다. 이는 "top-down" 접근법이라고 생각할 수 있는데, Agglomerative 방법과는 반대로 작동한다고 생각할 수 있습니다. 즉, 처음에 모든 데이터 포인트들이 하나의 군집에 속해 있다고 가정하고, 순차적으로 군집을 나눠주면서 진행되는 방식입니다.

여기서는 일반적으로 더 많이 사용되는 Agglomerative 방법에 대해서 설명하도록 하겠습니다. 문서들에 대해서 Agglomerative 방법의 위계적 군집 분석 알고리즘이 작동하는 방식은 아래와 같습니다.

왼쪽의 그림처럼 (하나의 문서로 구성된) 군집들이 특정한 공간 상에 위치해 있다고 가정을 하겠습니다. 위계적 군집 분석은 이러한 군집들 중에서 거리가 가장 가까운 (혹은 유사도가 큰) 2개의 군집끼리 묶어서 순차적으로 군집을 키워가는 방식입니다. **처음에 가장 가까운 거리**에 있는 두 개를 찾아서 하나의 군집으로 만들고, 그 다음에는 그 다음 가까운 거리에 있는 군집들끼리 묶어서 또 새로운 군집을 만듭니다.

이런 순서대로 군집의 수가 하나가 될때까지 동일한 과정을 반복합니다.

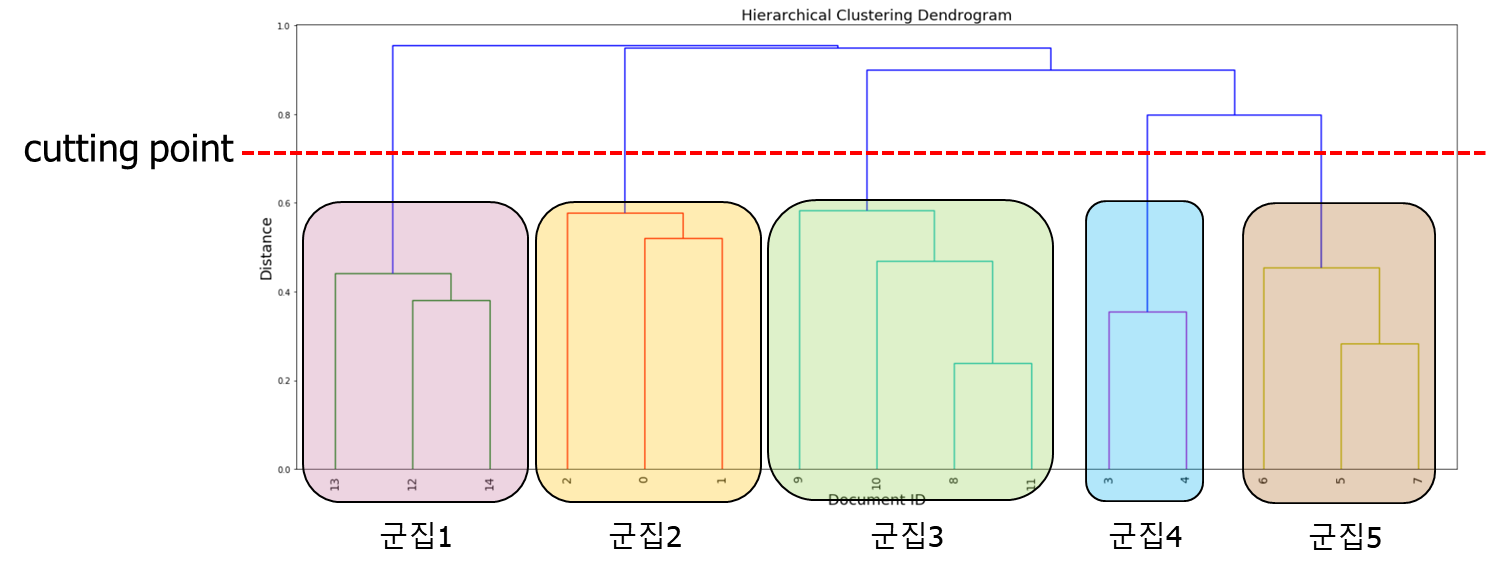
**그림변경 필요**

위계적 분석에서는 모든 점들이 다 연결될 때까지 알고리즘이 작동을 합니다. 그리고 그 결과를 덴드로그램 (dendrogram, 위 그림에서 왼쪽 아래) 이라는 그래프를 이용해서 시각화를 합니다. 덴드로그램을 이용해서 찾고자 하는 군집을 구분하게 됩니다.

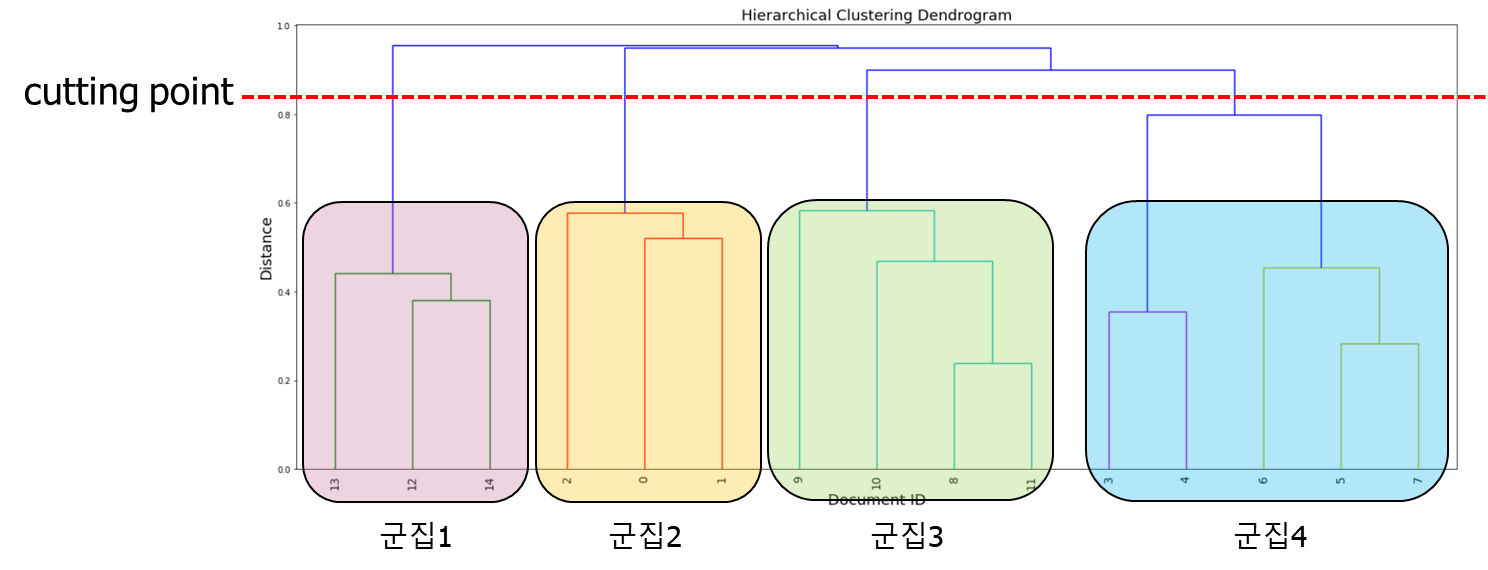
설명을 위해서 아래와 같은 새로운 덴드로그램을 살펴보도록 하겠습니다. 15개의 문서에 대해서 위계적 군집 분석을 통해서 아래와 같은 덴드로그램이 나왔다고 가정합니다.



위의 그림에서 가로축은 문서의 ID를 보여주고 있고, 세로축은 군집간의 거리를 보여주고 있습니다. 다시 말하지만 거리가 위계적 군집 분석 방법에서는 가까운 군집끼리 먼저 묶어주게 됩니다. 8번과 11번 문서 (즉, 군집)사이의 거리가 제일 짧아 (즉, 유사도가 제일 커서) 제일 먼저 묶인 것을 알 수 있습니다. 그 다음에는 5번과 7번 군집이 묶이게 됩니다. 위와 같은 덴드로그램에서 만약 우리가 찾고자 하는 최종 군집의 수가 5이라면 아래와 같이 군집간의 거리를 기준으로 cutting point를 지정할 수 있습니다. 그러면 아래 그림에서와 같이 전체의 데이터 포인트들이 5개의 군집으로 나눠지게 됩니다.



만약 cutting point를 아래와 같이 한다면 4개의 서로 다른 군집이 찾아지게 됩니다. 위의 그림과의 차이를 발견할 수 있겠나요? cutting point가 위의 그림에서 보다 더 높아진 것을 확인할 수 있습니다.



위계적 군집 분석의 경우에는 위와 같이 덴드로그램을 사용해서 적정수의 군집수를 상대적으로 쉽게 파악할 수 있습니다. 예를 들어, 위의 데이터 같은 경우에는 군집의수를 5으로 하는 것과 4로 하는 것 중에서 어떠한 것이 더 적절할까요? 이는 묶인 군집간의 거리를 통해서 확인할 수 있습니다. 군집을 4개로 하는 경우에는 네번째 군집에 존재하는 문서 간 (왼쪽의 2개의 문서와 오른쪽의 3개의 문서 간)의 거리가 상대적으로 큰 것을 알 수 있습니다. 따라서 이러한 경우에는 군집의 수를 5로 하는 것이 같은 군집내 존재하는 문서간의 유사도를 크게하는 더 나은 방법이라고 할 수 있습니다.

#### 군집을 묶어 주는 방법

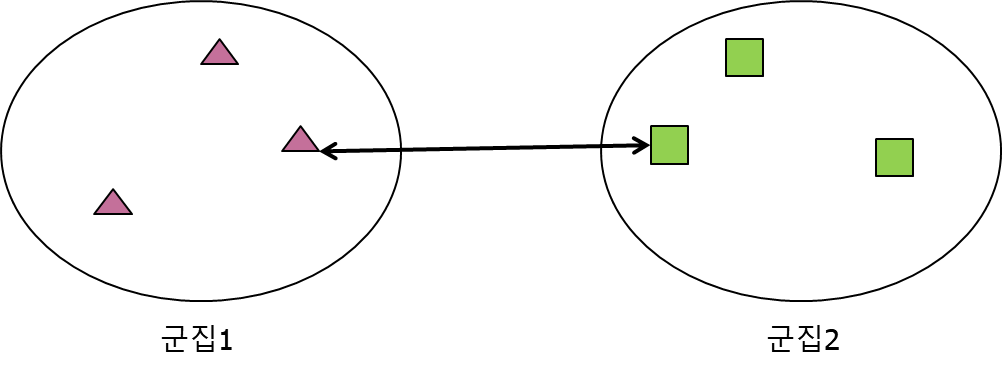
서로 다른 두 개의 군집을 묶어 주는 방법에는 크게 4가지가 존재합니다. 이중 3개는 군집에 속해 있는 데이터 포인트 간의 거리 지표를 사용하는 방법이고 나머지 하나는 군집 안에 존재하는 데이터 포인트들이 흩어진 정도, 즉, 분산을 이용하는 방법입니다.

**거리를 이용하는 방법**

거리를 이용하는 방법들은 모두 특정한 기준에 따라 두 군집 간에 거리를 계산한 다음에 거리가 가장 짧은 두 개의 군집을 하나의 군집으로 묶어 주는 방법입니다. 거리를 어떻게 측정하느냐에 따라서 3가지 방법으로 구분됩니다.

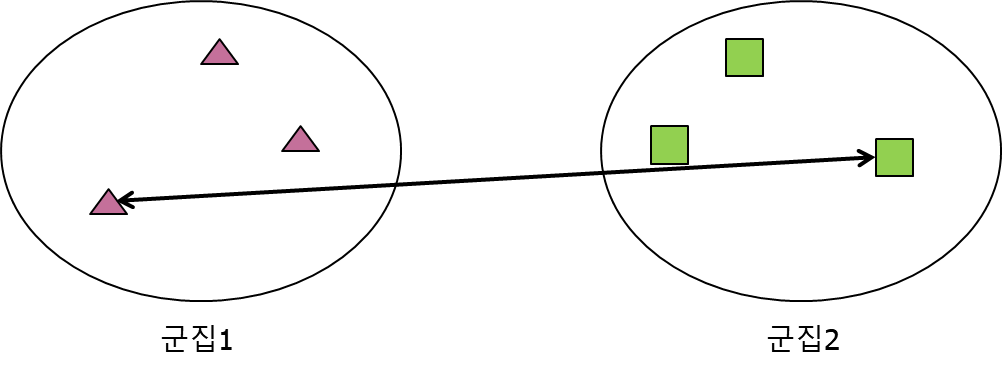
방법1) Single linkage

Single 연결 방법은 두 군집 간의 거리를 측정할 때 두 군집에 속한 데이터 포인트들 중에서 거리가 가장 가까운 두 데이터 포인트들 간의 거리를 두 군집 간의 거리로 사용하는 방법입니다 (아래 그림 참고). 두 데이터 포인트 간의 거리를 측정할 때는 주로 유클리디안 거리와 코사인 거리가 사용됩니다.



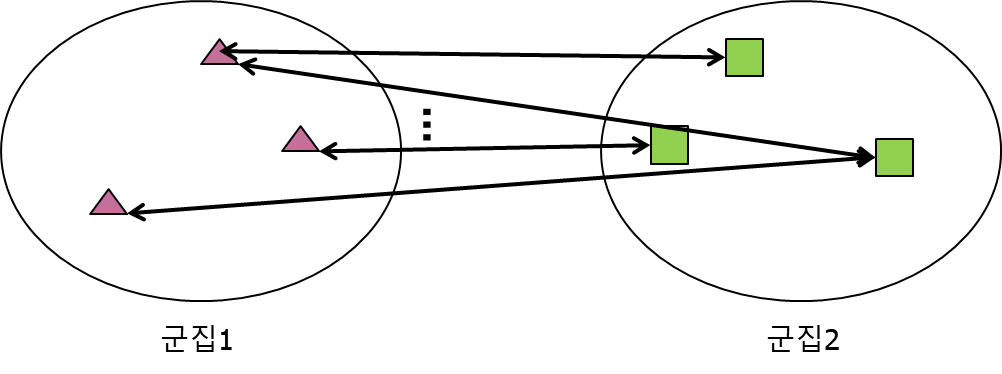
방법2) Complete linkage

Complete 연결 방법은 Single 연결 방법과는 반대로 작동합니다. 즉, 두 군집 간의 거리를 측정할 때 두 군집에 속해 있는 데이터 포인트들 중에서 거리가 가장 멀리 떨어져 있는 데이터 포인트 간의 거리를 두 군집 간의 거리로 사용합니다 (아래 그림 참고).



방법3) Average linkage

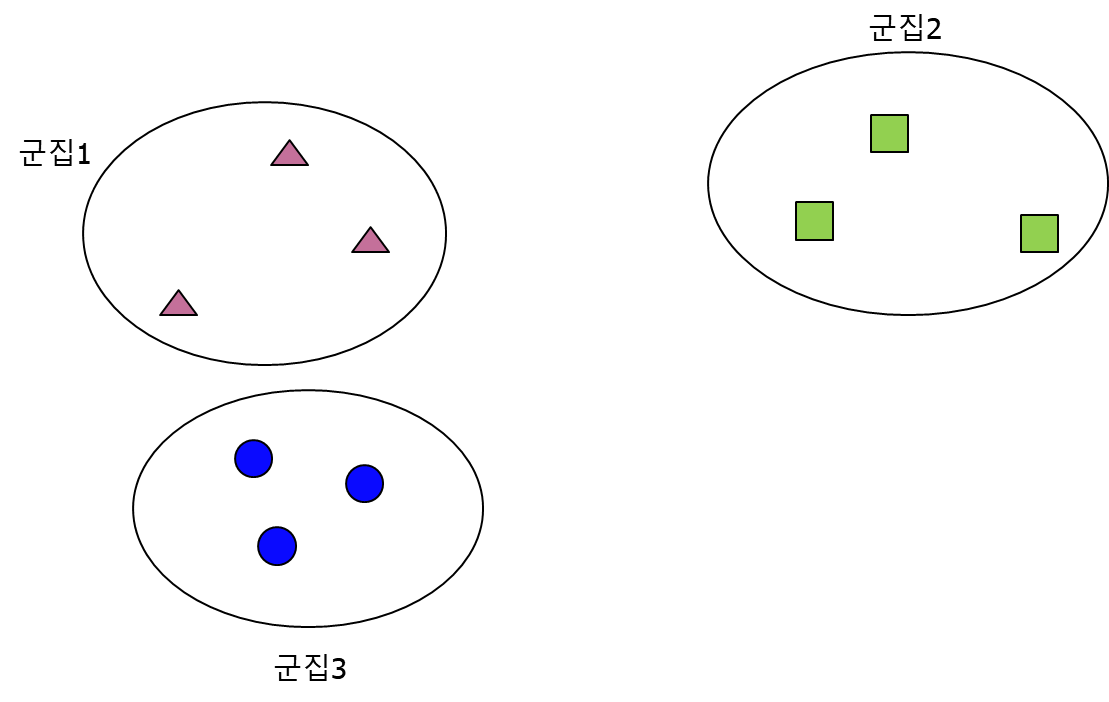
Average 연결 방법은 두 군집 간의 거리를 측정하기 위해서 각 군집에 속하는 데이터 포인트들 간의 평균 거리를 사용하는 방법입니다 (아래 그림 참고).



**분산을 사용하는 방법**

방법4) Ward linkage

Ward 연결 방법은 분산을 사용해서 두 군집을 묶어 주는 방법입니다 (이때 사용되는 분산을 Ward 분산이라고 합니다). 여러개의 군집들 중에서 서로 다른 2개의 군집을 묶었을 때 새롭게 생기는 군집에 속한 데이터 포인트들의 흩어진 정도, 즉, 분산이 제일 작게하는 군집들을 먼저 묶어주는 방법입니다. 예를 들어 다음과 같은 3개의 군집이 있다고 가정하겠습니다. 아래의 3개 군집들에 대해서 서로 두개를 묶을 수 있는 방법에는 3가지가 있습니다 (즉, 군집1-군집2 묶기, 군집2-군집3 묶기, 군집1-군집3 묶기). 이 3가지 방법 중에서 군집1-군집3을 우선적으로 묶어 줍니다. 왜냐하면, 군집1-군집3을 묶었을 때 생성되는 새로운 군집의 분산이 다른 방법으로 군집들을 묶었을 때 생기는 군집의 분산보다 작기 때문입니다 (즉, 데이터 포인트들이 흩어진 정도, 혹은 서로 떨어져 있는 정도가 더 작기 때문입니다).



#### 파이썬 코딩하기[[37]](#footnote-38)

여기서는 sklearn에서 제공되는 AgglomerativeClustering 클래스를 이용해서 Agglomerative 위계적 군집 분석을 해보겠습니다. 사용하는 데이터는 KMeans에서 다룬 15개의 신문기사 데이터입니다. Agglomerative 위계적 군집 분석을 하는 경우에도 기본적인 과정을 KMeans의 경우와 동일합니다. 즉, 전처리 과정을 통해서 DTM을 생성해야 합니다. 여기서는 앞에서 생성한 DTM\_TFIDF를 그대로 사용하겠습니다.

위계적 군집 분석을 위해서 아래와 같이 AgglomerativeClustering 클래스를 임포트 합니다.

from sklearn.cluster import AgglomerativeClustering

그리고 생성자 함수를 사용해서 객체를 생성합니다.[[38]](#footnote-39) 이 때, 생성자 함수의 주요한 파라미터로는 linkage, affinity, n\_clusters입니다. linkage는 군집간의 연결 방법에 대한 것으로 앞에서 설명한 4가지 값 중하나를 지정합니다. affinity는 벡터 간의 거리를 측정할 때 사용하는 방법입니다. 기본적으로 'euclidean' 또는 'cosine'을 사용합니다. n\_clusters는 찾고자 하는 군집의 수입니다. KMeans의 예에서와 동일하게 여기에서도 찾고자 하는 군집의 수를 5로 지정합니다. 아래의 코드에서는 linkage를 'average'로 그리고 affinity를 'cosine'으로 지정했습니다. 참고로, linkage='ward'로 지정한 경우에는 affinity 파라미터의 값을 지정할 수 없습니다. ward 방법은 벡터 간의 거리를 사용한 것이 아니라 분산을 사용한 것이기 때문에 그렇습니다.

agg = AgglomerativeClustering(linkage='average', affinity='cosine', n\_clusters=5)

그리고 아래와 같이 fit() 함수를 이용해서 군집화 분석을 합니다.

clusters\_agg = agg.fit(DTM\_TFIDF)

결과는 clusters\_agg.labels\_에 저장되어 있습니다. 결과가 의미하는 것은 KMeans의 결과와 동일합니다. 즉, 각 문서가 속한 군집의 ID가 됩니다.

print(clusters\_agg.labels\_)

[1 1 1 4 4 3 3 3 0 0 0 0 2 2 2]

적정의 군집 수를 찾기 위해서 덴드로그램을 사용할 수 있습니다. 덴드로그램은 아래의 코드를 사용해서 시각화합니다.

from matplotlib import pyplot as plt

from scipy.cluster.hierarchy import dendrogram, linkage

%matplotlib inline

np.set\_printoptions(precision=5, suppress=True)

Z = linkage(DTM\_TFIDF, 'average', metric='cosine')

plt.figure(figsize=(25, 10))

plt.title('Hierarchical Clustering Dendrogram', fontsize=18)

plt.xlabel('Document ID', fontsize=18)

plt.ylabel('Distance', fontsize=18)

dendrogram(

Z,

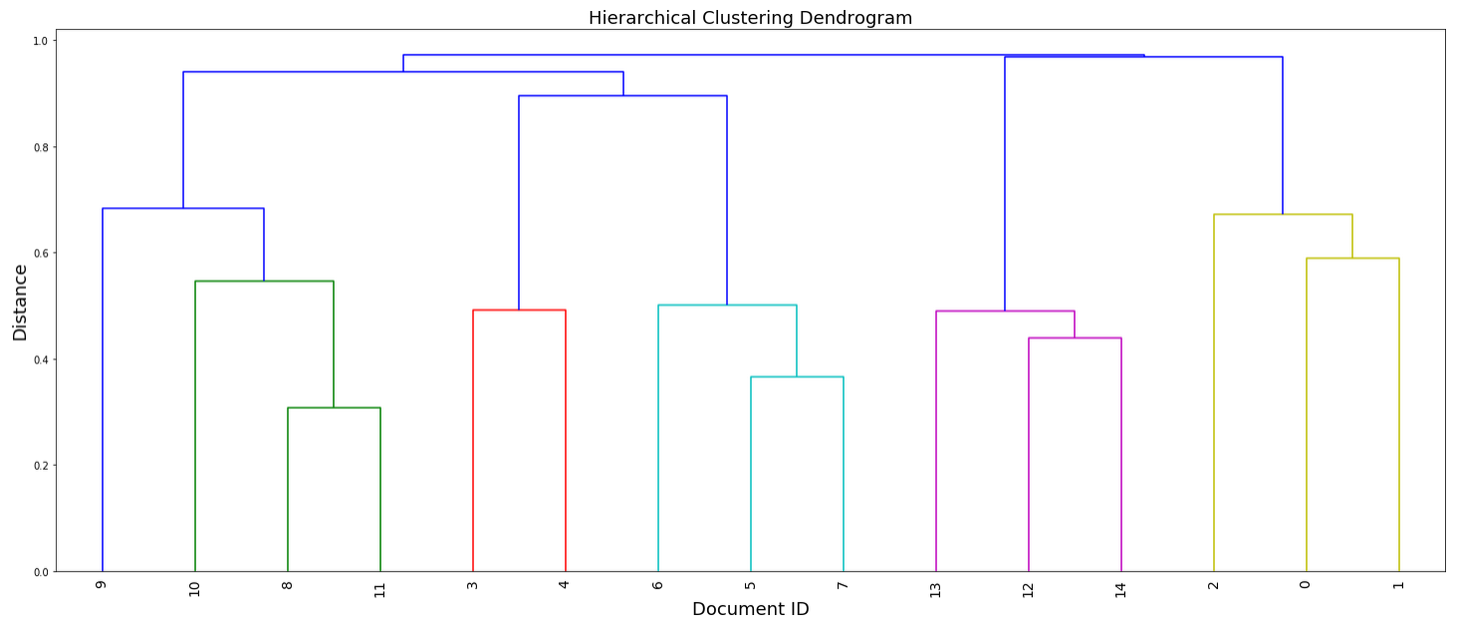
leaf\_rotation=90., # rotates the x axis labels

leaf\_font\_size=14., # font size for the x axis labels

)

plt.show()

덴드로그램 시각화 결과는 아래와 같습니다.

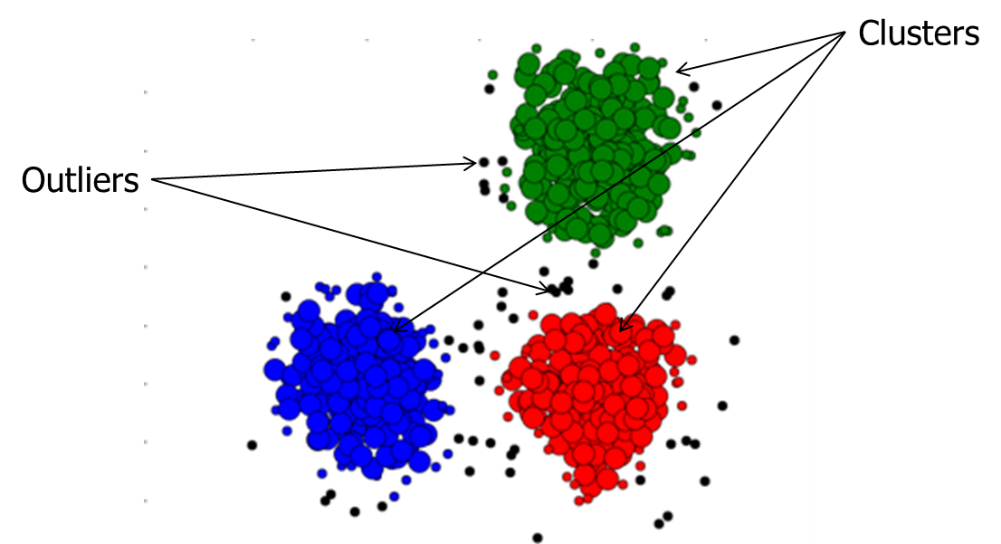


## DBSCAN (Density-based spatial clustering of applications with noise) 방법[[39]](#footnote-40)

DBSCAN 방법은 상대적으로 그 사용 빈도가 K-Means나 위계적 군집 분석 방법보다는 많지 않습니다. 하지만, 우리가 가지고 있는 텍스트 데이터의 특성에 따라서 그 성능이 좋을 수 있습니다. 따라서, 여기서 간략하게 그 작동 원리에 대해서 알아보도록 하겠습니다.

#### DBSCAN 설명

DBSCAN 방법은 데이터 포인트들이 밀집된 정도에 따라 (즉, density-based) 군집을 나누는 방법입니다. 즉, 데이터 포인트들이 상대적으로 더 밀집되어 있는 곳을 하나의 군집으로 지정하는 방식입니다. 이는 단순히 일정한 거리 내에 위치해 있는 데이터 포인트들을 서로 연결하면서 진행됩니다. 아래 그림에서와 같이 데이터 포인트들이 서로 밀집되어 있는 곳이 하나의 군집이 됩니다 (참고로 아무런 군집에도 속하지 않는 데이터 포인트들을 이상치(outlier)라고 합니다).



DBSCAN 알고리즘이 작동하는 방식을 설명하기 위해서 먼저 DBSCAN에서 사용되는 주요 용어에 대해서 알아보도록 하겠습니다.

- ε (epsilon /입실론/): 특정 데이터 포인트의 반경을 나타내는 값으로 하나의 데이터 포인트에서 다른 데이터 포인트로 연결 여부를 결정할 때 사용합니다. 즉, 특정한 데이터 포인트를 기준으로 ε 안에 다른 데이터 포인트가 있으면 해당 다른 데이터 포인트하고의 연결이 가능하고 그렇지 않으면 연결 불가능합니다.

데이터 포인트의 구분: DBSCAN에서는 각 데이터 포인트가 그 특성에 따라서 다음 4가지 중 하나로 구분됩니다.

① 코어 포인트(Core point): 어떠한 데이터 포인트 p로 부터 반경 ***ε*** 안에 적어도 특정수(보통 ***minPts*** 라고 표기됩니다) 이상의 데이터 포인트가 존재하는 경우 우리는 해당 데이터 포인트 p를 core point라고 합니다 (***ε*** 와***minPts***는 모두 사용자가 지정하는 하이퍼파라미터가 됩니다). core point만 다른 데이터 포인트로 연결할 수 있습니다.

② 직접적으로 도달 가능한 포인트 (directly reachable point): 만약 데이터 포인트 q가 core point p로 부터 ε 거리 안에 존재한다면 q는 p로 부터 직접적으로 도달 가능한 포인트가 됩니다.

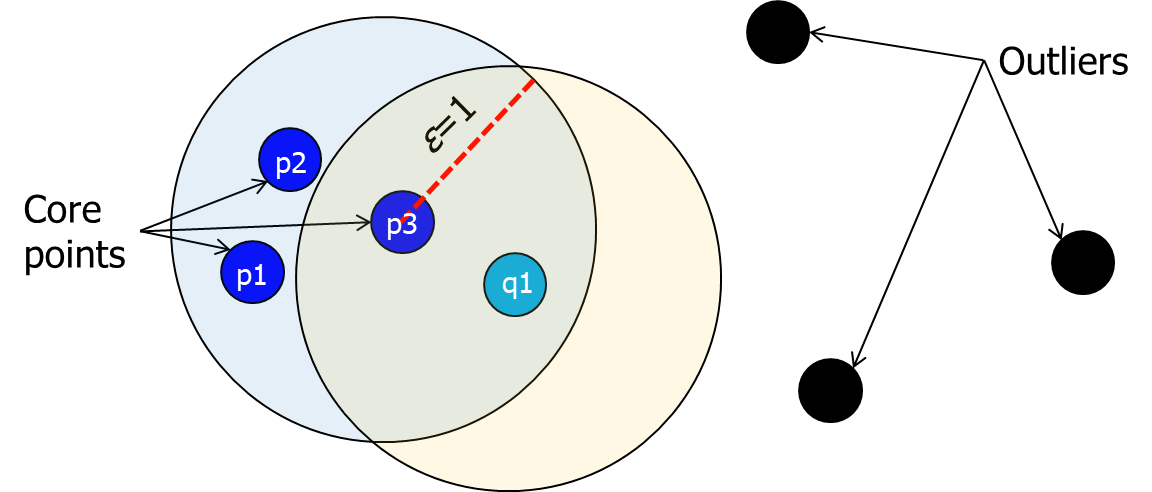
③ 도달 가능한 포인트 (reachable point): 만약 데이터 포인트 q가 코어 데이터 포인트 p로부터 직접 도달 가능한 (directly reachable) 포인트는 아니지만, 중간에 위치하는 다른 데이터 포인트들을 통해서 (간접적으로) 도달 가능한 경우에는 데이터 포인트 q를 데이터 포인트 p로부터 도달가능한 (reachable) 포인트라고 합니다 (아래 그림에서 q1은 p1으로부터 도달가능한 point가 됩니다). core point만이 다른 데이터 포인트에 도달(reach)할 수 있으므로, p와 q 중간에 있는 데이터 포인트는 모두 core points가 되어야 합니다 (아래 그림에서 p3).

④ 이상치 (outliers): 다른 어떠한 데이터 포인트로부터도 도달가능하지 못한 데이터 포인트들은 이상치가 됩니다. 또는 noise points라고 불립니다.

#### 군집의 구성

특정 데이터 포인트 p가 core point라면 해당 포인트는 해당 포인트로부터 도달 가능한 다른 데이터 포인트들과 함께 하나의 군집을 이룹니다. 하나의 군집에는 적어도 하나의 core point가 반드시 존재해야 합니다. core point가 아닌 데이터 포인트도 군집에 포함될 수 있는데 이러한 테이터 포인트(들)를 해당 군집의 edge라고 부릅니다.

예: ε=1, MinPts = 3 인 경우



위의 그림에서 p1, p2, p3는 core point들입니다. 왜냐하면 반경 1 (즉, ε=1)안에 자기 자신을 포함하여 적어도 3개의 데이터 포인트가 존재하기 때문입니다. q1은 core point가 되지 못합니다. q1은 p3로부터 직접적으로 도달가능한 point가 되고, p1으로부터는 도달가능한 point가 됩니다. 따라서, p1, p2, p3, q1의 하나의 군집을 이루게 됩니다. 검은색 데이터 포인트들은 core point로부터 도달가능하지 않으므로 같은 군집에 포함되지 않습니다. 그리고 군집을 이루지 못하기 때문에 이러한 포인트들을 이상치라고 합니다.

#### 파이썬 코딩 하기[[40]](#footnote-41)

DBSCAN을 이용한 군집 분석을 하기 위해서는 sklearn에서 제공되는 DBSCAN 클래스를 이용합니다. 여기서도 앞의 경우와 마찬가지로 15개의 신문기사로 구성된 데이터를 이용합니다. 앞서 만든 TF-IDF 기반의 DTM인 DTM\_TFIDF을 사용합니다. DBSCAN 클래스의 생성자 함수의 주요한 파라미터로는 입실론 (ε)을 의미하는 eps와 MinPts을 나타내는 min\_samples가 있습니다. 일반적으로 min\_samples의 값은 3으로 설정됩니다 (물론 여러분은 다른 값들도 사용해 보세요). esp의 값을 설정하기 위해서 각 문서 벡터하고 가장 가까운 문서 벡터하고의 거리 정보를 사용합니다. 이를 위해 아래와 같은 사용자 정의함수를 사용합니다.

# get distance with nearest neighbor

def nearest\_dist(a):

norms = []

for i in range(len(a)):

temp\_norms = []

for j in range(len(a)):

if i != j:

norm = np.linalg.norm(a[i]-a[j])

temp\_norms.append(norm)

norms.append(min(temp\_norms))

return norms

DTM\_TFIDF에 대해서 위의 함수를 적용하면 각 문서 벡터하고 가장 거리가 가까운 다른 문서 벡터하고의 거리가 계산됩니다.

short\_distances = nearest\_dist(DTM\_TFIDF)

print(short\_distances)

Out: [1.0862427819879066, 1.0862427819879066, 1.1509812879198542, 0.9912542627412413, 0.9912542627412413, 0.855206166281706, 0.9549035416762575, 0.855206166281706, 0.7840743895235609, 1.1454664510507953, 0.9986951838977298, 0.7840743895235609, 0.9367518539675339, 0.9638313983559299, 0.9367518539675339]

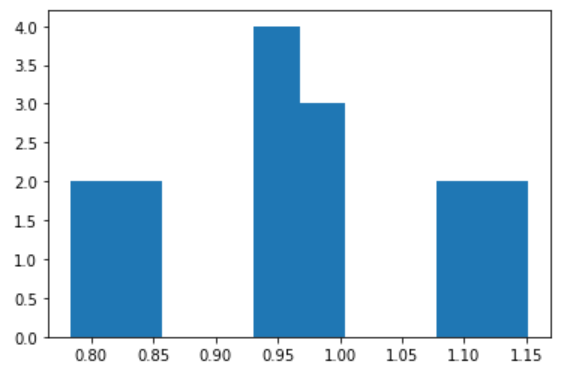
첫번째 문서 벡터와 가장 가까운 문서 벡터의 거리는 1.086 정도가 됩니다. 위의 값들의 분포를 확인해 보겠습니다.

import matplotlib.pyplot as plt

plt.hist(short\_distances)

plt.show()

아래와 같은 결과가 나옵니다.



최대값은 1.15 정도가 되고, 평균은 0.97 정도가 됩니다.

max\_dis = np.array(short\_distances).max()

print(max\_dis) # 1.1509812879198542

mean\_dis = np.array(short\_distances).mean()

print(mean\_dis) # 0.9680624514602977

여기서는 DBSCAN() 생성자 함수의 eps 파라미터의 값으로 위의 최대값과 평균을 각각 활용해 보도록 하겠습니다. 최대값과 평균 보다 약간 큰 값을 사용하게 됩니다.

# 최대값을 사용하는 경우

from sklearn.cluster import DBSCAN

dbscan1 = DBSCAN(eps=max\_dis+0.01, min\_samples=3)

clustering\_DBS\_max = dbscan1.fit(DTM\_TFIDF)

print(clustering\_DBS\_max)

Out: [ 0 0 0 -1 -1 1 1 1 2 2 2 2 3 3 3]

위의 결과는 앞의 경우와 동일한 의미를 갖습니다. 다만 -1 라고 하는 값이 보이는데 이는 DBSCAN에서의 outlier를 의미합니다. 즉, 어떠한 군집에도 속하지 않은 문서가 됩니다. 네번째와 다섯번째 문서를 제외하고는 KMeans 또는 위계적 군집분석의 결과와 동일한 것을 알 수 있습니다 (물론 군집 ID는 다를 수 있습니다).

이번에는 평균 거리를 이용해 보겠습니다.

dbscan2 = DBSCAN(eps=mean\_dis+0.01, min\_samples=3)

clustering\_DBS\_mean = dbscan2.fit(DTM\_TFIDF)

print(clustering\_DBS\_mean.labels\_)

Out: [-1 -1 -1 -1 -1 0 0 0 -1 -1 -1 -1 1 1 1]

평균 거리를 사용한 경우에는 outlier에 해당하는 문서가 더 많은 것을 확인할 수 있습니다. 더 보수적인 결과를 반환하는 것입니다.

## 가우시안 혼합 모형 (Gaussian Mixture Model, GMM)

# 텍스트 분류: 감성 분석의 경우

텍스트 분류는 종속변수가 범주형 변수인 경우를 의미

신문기사의 섹션 (정치, 경제, 사회 등), 신문기사를 작성한 신문사의 정치 성향 (보수, 진보 등), 문서의 감성 (Sentiment, 즉, 긍정, 부정) 등이 그러한 변수의 예입니다. 종속 변수가 범주형 변수이기 때문에 문제의 종류가 분류 문제가 되고, 분류 문제에 적용될 수 있는 다양한 알고리즘이 사용될 수 있습니다. 종속 변수가 무엇인지와 상관없이 알고리즘이 작동하는 방식이나 원리는 동일하기 때문에 여기서는 종속변수가 문서의 감성인 경우에 대해서 설명합니다.

감성분석 (sentiment analysis)은 또 다른 말로는 opinion mining이라고도 불립니다. 감성 분석은 하나의 문서를 해당 문서에 사용된 단어가 무엇이냐에 따라서 긍정 또는 부정의 문서로 구분하는 것을 의미합니다. 예를 들어, 특정 영화에 대해서 다음과 같은 두개의 영화평이 있다고 가정합시다 (감성분석에서는 하나의 영화평이 하나의 문서가 됩니다).

사람은 각 영화평을 읽고, 쉽게 첫번째 영화평은 해당 영화에 대한 긍정의 의미를 담고 있고, 두번째 영화평은 부정의 의미를 담고 있다고 알 수 있습니다. 따라서 ‘영화평1 = 긍정, 영화평2 = 부정’으로 구분할 수 있는 것이지요. 하지만, 문서의 양이 많은 경우에는 사람이 작업하는 것이 효율적이지 못합니다. 뿐만 아니라, 사람의 판단에는 주관이 작용하기 때문에 그 결과의 객관성이 떨어질 수 있습니다. 이러한 이유로 우리는 컴퓨터를 이용해서 감성분석을 하게됩니다 (특히, 문서의 양이 많은 경우). 컴퓨터를 사용한 이러한 감성 분석은 크게 두가지 방법을 통해서 이루어집니다. 하나는 기계학습 (Machine Learning)을 사용하는 것이고 다른 하나는 감성어 사전을 사용하는 것입니다. 최근에는 기계학습 알고리즘의 발달과 학습 데이터의 증가로 인해 많은 경우 감성 분석은 기계학습 기반의 방법들을 사용합니다. 여기서는 기계학습 기반의 방법을 중점적으로 설명하도록 하겠습니다. 감성어 사전 기반 방법은 간략하게 다룹니다.

영화평1: 이 영화 정말 재미있다.  
영화평2: 별점 1점도 아깝다.

감성분석은 지도학습 기반의 기계학습 알고리즘을 사용합니다. 지도학습은 정답과 힌트 간의 관계를 파악하기 위해 특정한 기계학습 알고리즘 (혹은 모형)을 선택하고, 해당 모형을 이용하여 정답과 힌트 정보가 모두 있는 학습 데이터(training data)를 사용하여 정답과 학습 간의 최적의 관계를 학습한 다음, 그 결과를 이용하여 정답이 없고 힌트 정보만 있는 새로운 데이터(unseen data)의 정답을 예측하는데 사용됩니다. 감성분석에서의 정답은 문서의 긍∙부정이 됩니다. 이러한 형태의 정답을 레이블 (label)이라고 하기도 합니다.

영화평에 대한 감성분석을 예로 들어보겠습니다. 영화평에 대한 감성분석에서는 영화평 하나가 하나의 문서가 되는 것입니다. 그리고 우리는 각 문서의 감성, 즉, 레이블이 긍정인지 부정인지를 파악하는 작업을 하게 됩니다. 지도학습 알고리즘을 사용해서 정답이 있는 학습 데이터를 학습하게 되는데, 학습 데이터에 존재하는 각 영화평에 대해서는 정답, 즉 긍∙부정의 레이블 정보가 달려 있습니다 (아래 참고).

학습 데이터의 예)

영화평 Label

돈이 아깝네요 부정  
강추입니다. 긍정  
또 보고 싶어요. 긍정

…

우리는 지도학습 모형을 이용하여 학습 데이터 존재하는 정답 (즉, 레이블)과 힌트 간의 관계를 파악하게 됩니다 (이러한 과정을 학습이라고 하는 것입니다). 보통의 지도학습 모형은 파라미터를 가지고 있습니다. 파라미터가 어떠한 값을 갖느냐에 따라서 모형의 형태가 달라지는데, 우리가 학습을 통해서 해야하는 것은 학습 데이터에 존재하는 정답과 힌트 간의 관계를 가장 잘 설명하는 모형의 파라미터 값을 찾는 것입니다. 그리고 그렇게 찾아진 파라미터 값을 사용해서 정답이 없는 새로운 데이터에 존재하는 영화평의 정답, 즉, 긍∙부정 레이블을 예측하게 됩니다. 감성분석에서 정답을 맞출 때 사용되는 정보는 단어 정보가 됩니다. 즉, 학습을 통해서 어떠한 단어들이 얼마나 사용되었는지에 따라 해당 영화평이 긍정일 확률과 부정일 확률이 어떻게 되는지를 계산하게 됩니다. 그리고 이러한 학습의 결과를 긍,부정의 레이블이 없는 새로운 영화평 데이터에 적용하게 됩니다. 그리고 (학습을 토대로 도출된 구체적인 파라미터값을 갖는 모형을 바탕으로) 새로운 데이터의 각 영화평에 대해서 어떠한 단어들이 얼마나 사용되었는지에 따라 해당 영화평이 긍정일 확률과 부정일 확률을 계산하게 되는 것이지요. 그리고 긍정과 부정 중에서 확률이 높은 것으로 해당 영화평의 감성을 결정하게 됩니다.

기계학습 알고리즘을 이용해서 수행되는 감성 분석의 전체 과정은 아래와 같습니다.

① 텍스트 데이터 준비

기계학습을 이용한 감성분석은 지도학습 알고리즘을 사용하기 때문에 정답이 있는 데이터가 준비되어야 합니다. 즉, 여러개의 문서와 각 문서의 감성이 긍정인지 부정인지에 대한 정보를 갖고 있는 데이터가 준비되어야 합니다. 이러한 데이터를 가지고, 학습과 평가를 할 수 있습니다. 뿐만 아니라, 학습을 통해 도출된 특정한 형태의 모형을 적용하기 위한 새로운 데이터 (unseen data)도 있어야 합니다. 우리가 궁극적으로 하고자 하는 것은 새로운 데이터에 존재하는 각 문서의 감성을 파악하는 것입니다.

② 텍스트 데이터 전처리

정답이 있는 텍스트 데이터를 준비했다면, 분석에 사용할 수 있도록 형태로 변환해야 합니다. 즉 텍스트 데이터의 전처리를 해야하고 전처리 과정을 통해 추출이된 최종 단어 정보를 사용하여 텍스트 데이터에 존재하는 각 문서를 벡터로 변환해야 합니다. 전처리 관련 섹션에서 설명한 것 처럼 전처리 과정을 통해서 우리는 일반적으로 불용어가 제거된 문서의 특성을 잘 대표하는 단어들을 추출하게 됩니다. 감성분석의 경우에는 최종적으로 선택된 단어들이 감성분석 관련하여 각 문서의 특성, 즉, 감성적인 특성을 잘 반영할 수 있는 단어들 이어야 합니다. 이렇게 분석에 사용되는 최종적으로 선택된 단어들을 features라고 부릅니다. 보통 전처리 과정을 통해서 우리는 불용어가 제거된 특정한 품사의 단어들을 선택하게 됩니다. 감성분석 관련해서는 감성을 나타내는데 사용되는 형용사와 부사의 단어들을 선택하는 것이 필요합니다.

③ Vectorization (DTM으로 변환)[[41]](#footnote-42)

전처리가 끝난 다음에는 각 문서를 최종적으로 선택된 단어들 정보를 사용하여 벡터로 변환해야 합니다. 이러한 과정을 벡터화 과정 (vectorization)이라고 합니다. 그리고 각 문서의 벡터 정보를 연결하여 전체 데이터를 하나의 행렬로 표현하게 됩니다. 이러한 행렬을 문서-단어 행렬 (Document-term matrix, DTM)이라고 합니다. 그런 다음 DTM을 이용해서 우리가 선택한 기계학습 알고리즘을 학습을 시키고 그 결과를 정답이 없는 새로운 데이터에 적용해서 각 문서가 긍정, 부정일 확률을 예측하게 됩니다.

④ 기계학습 모델을 학습  
이렇게 데이터가 준비되어 있으면, 이를 이용해서 기계학습 알고리즘을 학습 시키게 됩니다. 감성 분석, 좀더 일반적으로는 텍스트 분류에 사용될 수 있는 전통적인 기계학습 알고리즘으로는 나이브 베이즈(Naïve Bayes), 로지스틱 회귀, 결정 트리 (Decision Tree) 기반의 앙상블 (Ensemble) 방법들 (예, Random Forest, XGBoost, Light GBM 등), SVM (Support vector machine) 등의 알고리즘이 있습니다. 이외에 딥러닝 알고리즘도 사용될 수 있습니다. 딥러닝 알고리즘의 경우는 (일반적으로) 문서 (혹은 문장)에서 사용된 단어의 순서까지 고려하기 때문에 그 결과가 더 정확할 수 있습니다. 딥러닝 알고리즘에 대해서는 3부에서 자세히 다룹니다.

텍스트 분류에 적용될 수 있는 일부 기계학습 알고리즘 (예, 나이브 베이브, 로지스틱 회귀모형 등)은 확률을 기반으로 하고 있습니다. 따라서 해당 알고리즘들의 작동 원리를 보다 잘 이해하기 위해서는 확률의 기본 개념에 대해서 이해하고 있는 것이 필요합니다. 이와 관련해서 해당 알고리즘들에 대해 설명하기 전에 확률의 기본 개념에 대해 먼저 살펴보도록 하겠습니다 (본 책에서는 확률 기반의 알고리즘을 이해하는 필요한 필수적인 내용만 설명합니다).

## 로지스틱 회귀모형을 이용함 감성 분석

### 로지스틱 회귀모형 (Logistic regression model)

로지스틱 회귀모형은 일반적으로 종속변수가 취할 수 있는 값의 수가 한정되어 있는 범주형 변수 (categorical variable, 또는 명목변수 (nominal variable))인 문제에 적용됩니다. 종속변수가 취할 수 있는 값의 수가 두 개인 문제에 적용되는 로지스틱 회귀 모형을 이항(binomial) 로지스틱 회귀 모형이라고 하고 세 개 이상인 문제에 적용되는 모형을 다항(multinomial) 로지스틱 회귀 모형이라고 합니다. 이항 모형과 다항 모형의 경우 작동하는 원리는 유사하기 때문에 여기서는 종속변수가 취할 수 있는 값이 두 개, 즉, 긍정과 부정인 경우에 대해서 설명하도록 하겠습니다.

종속변수(Y)가 취할 수 있는 값이 두 개인 경우 보통 0과 1로 표현합니다 (-1과 1로 표현하는 경우도 있습니다). 감성분석의 경우에는 긍정을 1로 부정을 0으로 표현하게 됩니다. 이러한 경우 로지스틱 회귀 모형에서의 Y=1일 확률은 아래와 같이 정의됩니다. 오른쪽 변에 존재하는 함수가 표준 로지스틱 분포에 대한 누적분포함수이기 때문에 해당 모형을 로지스틱 회귀 모형이라고 합니다.

여기에서 으로 표현됩니다. 는 입력값, 즉 독립변수 (기계학습에서는 특성 정보라고 표현합니다)를 의미합니다. 그리고 는 각 독립변수에 대한 파라미터이고, 이는 Y값을 결정하는데 각 독립변수가 얼마나 중요한 역할을 하는지를 나타내는 값이라고 볼 수 있습니다. 텍스트 분석의 경우 각 독립변수()는 각 단어가 됩니다. 그리고 가 취하는 값은 일반적으로 각 단어가 문서에 대해서 갖는 사용 빈도(frequency) 또는 TF-IDF 값이 됩니다. Y가 취하는 값이 0과 1 밖에 없기 때문에 Y가 0의 값을 취할 확률 (즉, )은 가 됩니다. 는 조건부 확률로 Y=1일 확률이 독립변수들()이 어떠한 값을 취하느냐에 따라 달라진다는 것을 의미합니다.

7 장에서 다룬 선형회귀모형과 마찬가지로 로지스틱 회귀 모형도 하나의 수학적 모형입니다. 그리고 이 모형이 갖는 파라미터가 되고, 이는 모형의 종속변수 와 독립변수들 간의 관계를 정의하는 역할을 합니다. 따라서 학습 데이터를 학습해서 비용함수를 최소화하는 파라미터의 값들을 찾게 됩니다. 로지스틱 회귀 모형은 분류 문제 (classification problem)에 적용되기 때문에 교차 엔트로피 비용함수를 사용하게 됩니다.

파라미터 값의 의미

로지스틱 회귀 모형은 비선형 모형 (즉, 모형이 파라미터에 대해서 비선형)이기 때문에 파라미터의 값이 선형회귀모형에서의 기울기 의미를 갖지 않습니다. 즉, 선형회귀모형에서의 파라미터 값은 해당 독립변수의 값이 1만큼 증가할 때 달라지는 종속변수의 값을 의미했습니다. 하지만, 로지스틱 회귀 모형의 파라미터는 그러한 의미를 갖지 않는 것입니다. 파라미터 값을 통해서 우리가 파악할 수 있는 것은 크게 두가지 입니다. 첫 번째는 해당 독립변수와 모형의 종속변수인 가 갖는 관계의 방향입니다. 예를 들어, 인 경우에는 해당 독립변수인 과 이 양의 관계를 갖는다는 것을 (즉, 값이 증가하면 Y가 1의 값을 갖을 확률도 증가한다는 것을) 의미합니다. 반대로 인 경우에는 해당 독립변수인 과 이 음의 관계를 갖는다는 것을 의미합니다.

그리고 독립변수의 값들이 정규화 혹은 표준화 되어 있는 경우, 파라미터의 절대값의 크기는 독립변수가 종속변수를 설명하는 정도를 나타내게 됩니다. 예를 들어, 이고 인 경우 의 절대값이 더 크기 때문에 해당 독립변수이 가 종속변수를 설명하는 정도가 보다 크다는 것을 의미합니다.

예를 들어, 감성분석을 수행해서 다음과 같은 파라미터의 값이 도출되었다고 가정합니다.

위의 경우, ‘노잼’이라고 하는 단어 (텍스트 분석에서는 하나의 단어가 독립변수 혹은 특성 정보로 간주된다고 하였습니다)의 파라미터 값이 -3이고 ‘최고’라는 단어의 파라미터 값이 5가 나왔습니다. 즉, ‘노잼’ 단어는 Y=1일 확률 (즉, 문서의 감성이 긍정일 확률)과 음의 관계를 갖고, ‘최고’ 단어는 양의 관계를 갖는다는 것을 의미합니다. 그리고 ‘최고’ 단어 파라미터의 절대값이 ‘노잼’ 단어 파라미터 절대값 보다 더 크기 때문에 ‘최고’ 단어가 종속변수인 문서의 감성을 설명하는데 혹은 예측하는데 있어 더 큰 역할을 한다고 해석할 수 있습니다.

### 비용함수: 교차 엔트로피

종속변수가 취할 수 있는 값이 0과 1인 경우 (즉, ), 교차 엔트로피 (Cross Entropy) 비용함수는 다음과 같이 정의 됩니다.

위 식에서 은 학습 데이터에 존재하는 전체 관측치의 수를 나타내고, 는 번째 관측치의 실제 종속변수 값, 그리고 는 모형을 통해서 예측되는 일 확률을 의미합니다. 로지스틱 모형을 사용하는 경우, 로지스틱 모형을 이용해서 이 예측되는 것입니다. 위 비용함수는 종속변수의 실제값에 대한 확률을 제대로 예측할수록 그 값이 작아지고, 반대로 예측을 잘못 할수록 그 값이 커지게 됩니다. 따라서 위의 값을 최소로 하는 파라미터의 값을 찾아야 합니다.

위의 비용함수는 두 가지 방법을 이용해서 도출될 수 있습니다. 하나는 최대우도추정 (maximum likelihood estimation) 방법이고 다른 하나는 정보이론의 엔트로피 개념을 바탕으로 한 방법입니다.

#### 비용함수 도출: 최대우도추정 기반

어떠한 변수가 취할 수 있는 값이 두 개인 경우, 해당 변수는 베르누이 (Bernoulli) 확률 분포를 따른다고 할 수 있습니다. 베르누이 확률 분포의 확률질량함수는 아래와 같습니다.

여기에서 을 의미합니다.

로지스틱 회귀 모형의 경우에는

이 되는 것입니다. 여기서 입니다. 이를 위 확률질량함수 식에 대입하면 이 됩니다.

우리가 가지고 있는 데이터의 확률 (혹은 데이터가 발견될 확률)을 우도 (likelihood, 또는 가능도)라고 합니다. 관측치가 서로 독립이라고 하는 가정하는 경우에, 즉 번째 관측치의 값이 1일 확률이 다른 관측치가 취하는 값에 의해서 영향을 받지 않는다라는 가정하는 경우, N개의 데이터 포인트에 대한 우도는 다음과 같이 정의됩니다.[[42]](#footnote-43)

각 관측치의 종속변수가 특정한 값을 갖을 확률을 곱한 것이 됩니다. 이는 간단히 다음과 같이 표현됩니다.

(Pi, 파이)는 곱하기를 의미합니다. 우도는 파라미터, 즉 에 대한 함수가 됩니다. 우리가 가지고 있는 데이터에 대한 우도 ()의 값을 최대로 하는 파라미터의 값을 찾게 됩니다. 이러한 방법을 통해서 파라미터의 값을 찾는 것을 최대우도추정 (maximum likelihood estimation)이라고 합니다. 그런데, 곱하기는 미분 등의 계산을 하는 것이 쉽지 않기 때문에 앞에 자연로그를 붙여 줍니다. 이를 로그 우도(log-likelihood)라고 합니다.

두 수의 곱하기에 로그를 취하면 더하기로 표현됩니다 (즉, ). 따라서 위의 우도함수에 자연로그를 취하면 아래와 같이 됩니다.

이를 로그 우도 함수라고 합니다. 여기에서 가 베르누이 변수이기 때문에

같습니다. 이를 로그 우도 함수 ()에 대입하면 아래 식을 얻습니다.

여기에서 를 의미합니다. 는 가 1의 값을 갖을 확률이 해당 관측치의 독립변수 정보()와 모형의 파라미터 값()에 영향을 받는다는 것을 나타내는 조건부 확률입니다. 를 위식에 대입하면 아래와 같습니다.

여기에서 로지스틱 회귀 모형을 사용하는 경우,

가 됩니다.

위의 로그 우도 함수 ()의 값을 최대화하는 파라미터(즉, )의 값을 찾게 됩니다. 하지만, 기계학습에서 사용하는 로지스틱 회귀 모형은 위의 로그우도함수를 목적 함수로 직접 사용하지 않고, 앞에 마이너스(–)를 붙여서 비용함수로 만듭니다. 즉, 아래와 같은 비용함수를 사용하게 되는 것입니다.

위의 비용함수를 최소화하는 것은 결국 로그우도함수를 최대화 하는 것과 같습니다.

#### 비용함수 도출: 교차 엔트로피 기반

교차 엔트로피 비용함수는 정보이론의 교차 엔트로피(cross entropy)개념을 가지고도 설명할 수 있습니다. 그래서 해당 비용함수를 교차 엔트로피 비용함수라고 합니다. 이를 설명하기 위해서 엔트로피의 개념을 먼저 설명하겠습니다.

**엔트로피 (entropy)**

엔트로피라는 개념은 정보 이론 (information theory)에서 사용되는 개념입니다.[[43]](#footnote-44) 엔트로피는 변수의 불확실성 (uncertainty)을 의미합니다. 확률분포 p를 갖는 어떠한 변수 X에 대해서 엔트로피는 다음과 같이 정의됩니다. 여기서는 설명을 위해서 변수 X가 이산변수, 즉, 한정된 수(K개)의 값을 취하는 변수라고 가정합니다.

해당 값은 X가 특정한 값 하나를 가질 확률이 1인[[44]](#footnote-45) 경우에 최소가 됩니다. 그 때의 값은 0이 됩니다 (log1의 값은 0입니다). 반대로 각 값을 가질 확률이 동일한 경우, 즉 위의 경우는 p(X=k) = 1/K 인 경우 엔트로피 값이 최대가 됩니다. 이러한 경우 불확실성이 제일 크다고 생각할 수 있습니다.

**교차 엔트로피 (cross entropy)**

엔트로피가 하나의 변수 혹은 해당 변수가 갖는 확률 분포의 불확실성을 의미한다면, 교차 엔트로피는 하나의 변수 (X)가 가질 수 있는 서로 다른 분포들 (예, 와 )간의 차이를 의미합니다. 교차 엔트로피는 아래와 같이 정의 됩니다.

여기서 X는 1, 2, …, K 값을 취할 수 있습니다. 두 분포()의 차이가 클수록 값이 커지게 됩니다.

로지스틱 회귀 모형에서는 를 종속변수()의 실제값()을 그리고 를 모형을 통해서 예측된 값()을 이용해서 나타냅니다. 종속변수가 취할 수 있는 값이 0과 1인 경우, 그리고 가 됩니다. 종속변수가 취하는 값이 두 개인 경우, 하나의 관측치에 대한 교차 엔트로피는 다음과 같이 표현됩니다.

값이 1인 경우 , 경우는 가 교차 엔트로피가 되는 것입니다. 위의 식에서 은 P(y=1)을 의미합니다. 즉, 실제 값과 모형을 통해 예측된 확률 간 차이가 크게 날수록 교차 엔트로피의 값이 커지는 것입니다. 즉, 비용함수를 최소화 한다는 것은 실제 값에 대해서 예측을 되도록 정확하게 한다는 것을 의미합니다.

따라서 전체 관측치에 대한 교차 엔트로피는

가 됩니다.

### 파이썬 코딩

#### 영화평 데이터

여기서는 영화평 데이터를 이용해서 감성 분석을 수행해 보도록 하겠습니다. 전체 영화평 데이터는 2016\_movie\_reviews.txt 파일에 저장되어 있습니다. 해당 데이터는 2016년도에 국내에서 개봉된 영화들 중에서 매출액 기준 상위 300개의 영화들에 대한 네이버 영화평 데이터입니다. 해당 파일은 용량이 커서 전처리 및 학습에 시간이 많이 소요되기 때문에 여기서는 설명을 위해서 원본 데이터의 일부를 사용하도록 하겠습니다. 일부의 영화평 데이터는 2016\_movie\_reviews\_part.txt 파일에 저장되어 있습니다 (해당 파일에 저장된 영화평의 수는 22,228개입니다).

#### 전처리하기

여기서는 로지스틱 회귀 모형을 통해서, 2016년도에 한국에서 개봉이된 300개의 영화에 대한 영화평 데이터를 가지고 감성 분석을 해보도록 하겠습니다. 파이썬 코드는 LR\_sentiment.ipynb 파일을 참고하면 됩니다.

파이썬에서 로지스틱 회귀 모형을 이용한 기계학습을 하기 위해서는 다음과 같이 sklearn 모듈에서LogisticRegression 를 import 합니다.

from sklearn.linear\_model import LogisticRegression

그리고, 미리 준비된 ‘2016\_filtered\_review.txt’ 파일을 사용합니다. 해당 파일에는 다음과 같은 내용이 포함되어 있습니다.

140693 아니 딴 그렇 비 비탄 총 대체 왜 들 온겨 7

첫번째 값은 영화의 id이며 두번째는 해당 영화에 대한 특정 영화평이고 마지막 값 (7)은 해당 영화평의 평점입니다. 영화평은 전처리 과정을 통해서 특정 품사를 갖는 단어들만 추출해 놓은 것입니다. 원본 데이터는 ‘2016\_reviews.txt’ 파일에 저장되어 있고, 전처리는 ‘reviews\_preprocessing.py’ 파일을 사용해서 했습니다.

로지스틱 회귀을 사용해서 분석을 하기 위해서는 일단 위의 데이터를 이용하여 학습 데이터를 구축해야 합니다. 감성분석의 경우는 종속변수가 긍정 또는 부정 두가지 값 중에 하나의 값을 갖습니다. 하지만, 우리가 지금 가지고 있는 데이터의 종속변수는 1 ~ 10 사이의 자연수를 취합니다. 따라서 이러한 평점 정보를 긍/부정의 레이블로 변환하는 것이 필요합니다. 즉, 평점이 몇 점 이상이면 해당 영화평을 긍정으로 반대로 평점이 몇 점 이하이면 해당 영화평을 부정으로 레이블링 하는 작업이 필요한 것입니다.

여기서는 8점 이상이면 긍정 (1), 4점 이하이면 부정 (0)로 레이블링을 했습니다. 나머지 평점을 갖는 영화평들은 학습 데이터로 사용하지 않았습니다. 그 이유는 평점이 4 ~ 8 사인 영화평들에 사용된 단어들은 긍정 또는 부정을 명확하게 표현하지 않거나, 긍정 또는 부정을 나타내는 단어들이 모두 포함되어 있어 학습의 목적으로 적합하지 않을 수 있기 때문입니다. 하지만, 평점 기준은 데이터의 특성 또는 학습의 성능에 맞게 여러분들이 자의적으로 선택을 할 수 있습니다. 여기서는 4점과 8점을 선택했지만, 여러분들은 다른 점수 (예, 3점 또는 9점 등)를 레이블링 기준으로 사용할 수 있는 것입니다.

**학습 데이터와 평가 데이터로 구분하기**

우리가 사용하는 모형의 성능을 평가하기 위해서 정답이 있는 데이터의 일부를 평가 데이터 (test data)로 사용해야 합니다. 그리고 전체 정답이 있는 영화평 데이터 중에서 80%는 학습 데이터 (training data)로 나머지 20% 는 평가 데이터 (test data)로 사용하였습니다. 이를 위해서 sklearn에서 제공되는 train\_test\_split() 함수를 사용하였습니다.

from sklearn.model\_selection import train\_test\_split

train\_texts, test\_texts, train\_labels, test\_labels = train\_test\_split(filtered\_texts, filtered\_labels, test\_size=0.2)

train\_test\_split() 함수의 첫 번째 인자로는 영화평 정보가 두 번째 인자로는 영화평에 대한 긍/부정 레이블 정보가 제공되었습니다. 그리고 test\_size라는 파라미터를 통해서 평가 데이터의 비중을 설정하였습니다. 0.2는 전체 데이터 중에서 20%를 평가 데이터로 사용한다는 뜻입니다. 이렇게 하면 train\_test\_split() 함수는 학습 데이터와 평가 데이터를 구분해서 반환합니다. 학습 데이터의 영화평 정보는 train\_texts에, 정답인 레이블 정보는 train\_labels 변수에 저장되어 있습니다. 평가 데이터의 영화평 정보는 test\_texts에, 정답인 레이블 정보는 test\_labels 변수에 저장되어 있습니다.

우리는 학습 데이터 (즉, train\_texts와 train\_labels)를 사용해서 우리의 로지스틱 회귀 모형을 학습 시킵니다. 즉, 모형이 갖고 있는 파라미터의 최적값을 찾는 것입니다. 즉, 위에서 설명한 로지스틱 회귀 모형의 비용함수를 최소화하는 파라미터의 값들을 찾게 됩니다.

sklearn에서 제공되는 LogisticRegression 클래스를 이용해서 학습 시키기 위해서 텍스트 형태의 영화평 정보를 벡터로 변환합니다. 이를 위해서는 sklearn에서 제공되는 CountVectorizer 또는 TfidfVectorizer 클래스를 사용할 수 있습니다. CountVectorizer 클래스는 단어들의 사용 빈도 (즉, term frequency 정보)를 사용해서, TfidfVectorizer 클래스는 단어들의 TF-IDF 정보를 사용해서 각 영화평을 벡터로 변환합니다 (자세한 방법은 문서의 벡터화 섹션을 참고하세요). 여기서 여러분들이 주의해야하는 것은 영화평을 벡터화할 때 학습 데이터의 영화평과 평가 데이터의 영화평을 구분해서 벡터화 작업을 해야한다는 것입니다. 학습 데이터의 영화평들을 벡터화할 때 사용한 단어정보를 이용해서 평가 데이터의 영화평들도 벡터화를 해야 하는 것입니다. 이를 위해서 일단 먼저 학습 데이터의 영화평들을 벡터화하고, 학습 데이터를 벡터화할 때 사용한 정보를 이용해서 평가 데이터의 영화평도 벡터화 합니다 (아래 코드 참고). 이렇게 하면 학습 데이터에서 사용된 단어들만을 가지고 평가 데이터의 영화평들을 벡터화 하게 됩니다. 이렇게 해야지만 학습에서 사용된 단어들만을 이용해서 우리의 모형이 평가 데이터에서 어느 정도의 성능을 내는지 평가할 수 있습니다.

tf\_vectorizer, train\_tf\_features = tf\_extractor(train\_texts) # TF 정보를 이용해서 벡터화 합니다.

test\_tf\_features = tf\_vectorizer.transform(test\_texts)

우리는 독립변수 정보 (즉, feature 정보 또는 단어들 정보)와 종속변수 (즉, 긍/부정 레이블 정보)에 대한 학습 데이터 즉, train\_tf\_features와 train\_labels를 이용해서 단어들과 영화평이 긍정 또는 부정일 확률과의 관계를 학습하게 됩니다. 학습을 위해서 LogisticRegression 클래스에서 제공되는 fit() 함수를 사용합니다. 이를 위해서 먼저 LogisticRegression클래스의 객체를 만듭니다. 객체를 만들 때는 아래와 같이 LogisticRegression() 생성자 함수를 사용합니다. 해당 생성자 함수는 중요한 몇 개의 파라미터를 갖습니다. 그중 첫번째가 penalty 파라미터 입니다. 즉, regularization 방법으로 어떠한 방법을 사용할 것인지를 정하는 목적으로 사용됩니다. penaly 파라미터가 취할 수 있는 값은 4가지가 있습니다. 즉, {‘l1’, ‘l2’, ‘elasticnet’, ‘none’} 입니다. L1 방법을 사용하고자 하는 경우에는 'l1'을, L2 방법을 사용하고자 하는 경우에는 'l2'를, 그리고 L1과 L2를 같이 혼용해서 사용하고자 하는 경우에는 'elasticnet' 을 선택합니다. 아래 코드에서는 L1 (Lasso) 방식을 사용하여 객체를 생성하고 있습니다. 그리고 그 다음으로 설정해줘야 하는 파라미터가 C가 됩니다. C는 regularization strength의 역수가 됩니다. 즉, C = 1/ λ가 됩니다. 이는 λ=1/C를 의미함으로, C의 값이 작을수록 pentaly의 정도가 커지는 것입니다. penalty를 많이 준다는 뜻은 L1 같은 경우는 feature의 수를 그만큼 많이 줄인다는 뜻이고, L2인 경우는 weight 값을 더 0에 가깝게 한다는 뜻입니다. C는 사용자가 그 값을 설정해야 하는 하이퍼 파라미터입니다. 마지막으로 살펴봐야하는 파라미터는 solver라는 파라미터입니다. 이는 비용함수의 최소값을 어떠한 방법으로 찾을 것인지를 지정하는 역할의 파라미터입니다. 어떠한 solver를 사용하는지는 regularization 방법에 따라 달라집니다. L1 방법인 경우에는 saga solver를 사용합니다. 이는 경사하강법의 한 종류라고 생각하면 됩니다.

lr = LogisticRegression(C=0.1, penalty='l1', solver=‘saga’)

다음과 같이 fit()이라는 함수를 사용해서 학습데이터에 대해서 (즉, train\_tf\_features와 train\_labels에 대해서) 학습을 시킵니다. 학습을 통해서 각 영화평에 사용된 단어들과 영화평의 레이블 (즉, 긍/부정)의 관계를 가장 잘 설명하는 로지스틱 회귀 모형이 갖는 파라미터의 값이 결정됩니다.

lr.fit(train\_tf\_features, train\_labels)

train\_tf\_features는 train\_texts를 tf기반의 DTM으로 변환한 것입니다.

모형의 성능을 평가하기 위해서 학습에 사용되지 않은 (하지만 정답이 존재하는) 평가 데이터에 학습의 결과를 평가합니다. 이를 위해서 아래와 같이 평가 데이터에 존재하는 독립변수 정보 (즉, 영화평 정보)를 사용해서 영화평의 긍/부정을 예측합니다. 그리고 그 모형을 통해서 예측된 값과 평가 데이터에 존재하는 각 영화평에 대한 실제의 긍∙부정 값과 비교를 해서 우리가 학습시킨 모형의 성능을 아래와 같이 평가합니다. 여기서는 accuracy\_score()라는 함수를 사용하였습니다. 해당 함수는 모형의 정확도를 계산합니다.

pred\_labels = lr.predict(test\_tf\_features)

from sklearn.metrics import accuracy\_score

print('Accuracy: %.2f' % accuracy\_score(test\_labels, pred\_labels))

위 코드를 실행하면 accuracy\_score(test\_labels, pred\_labels) 함수는 0.93의 값을 반환합니다. 즉, 평가 데이터에 존재하는 전체의 영화평들 중에서 93%를 정확하게 맞혔다는 것을 의미합니다.

이번에는 L2 regularization 방법을 사용해 보도록 하겠습니다. 아래와 같이 생성자 함수를 호출합니다. penalty 파라미터의 값으로 이번에는 l2를 입력하여 L2 regularization 방법을 사용하게 했습니다. 그리고 L2 regularization에 대한 solver는 'sag'가 됩니다. 이 solver도 경사하강법의 한 종류라고 생각하면 됩니다.

lr2 = LogisticRegression(C=0.1, penalty='l2', solver='sag')

나머지 과정은 위와 동일합니다.

**각 단어의 긍/부정 정도 파악하기**

각 feature (즉, 각 단어)의 파라미터 값을 통해서 단어들 중에서 상대적으로 긍정적인 역할을 하는 단어와 부정적인 역할을 단어들의 목록을 파악할 수 있습니다. 이를 위해서 다음과 같이 학습의 결과로 얻어진 파라미터의 값들을 확보합니다.

coefficients = lr.coef\_.tolist()

coef\_.tolist()는 다음과 같은 결과를 반환합니다.

[(8402, 3.2834260607516965), (50619, 2.8758307801962313), (49991, 2.793400720353563), (50012, 2.7417887272975565), (50561, 2.731604141685505), …]

즉, 각 원소가 두개의 서로 다른 원소를 저장하고 있는 튜플이 됩니다. 예를 들어, 첫번째 튜플을 살펴보면 아래와 같습니다.

(8402, 3.2834260607516965)

첫번째 원소는 (즉, 8402)는 단어의 ID가 되며, 두번째 원소는 해당 단어의 파라미터 값이 됩니다. 해당 파라미터의 값이 클수록 긍정적인 역할을 하는 단어라고 생각할 수 있습니다. 즉, 해당 단어가 출현하는 영화평의 경우 긍정의 영화평일 확률이 높아지는 것입니다.

아래의 코드를 사용하여 상위 긍정단어 50개와 부정단어 50개를 확인해 볼 수 있습니다.

sorted\_coefficients = sorted(enumerate(coefficients[0]), key=lambda x:x[1], reverse=True) # 파라미터 값의 크기 순으로 정렬하기

for word, coef in sorted\_coefficients[:50]:

print('{0:} ({1:.3f})'.format(vocablist[word], coef))

# print top 50 negative words

for word, coef in sorted\_coefficients[-50:]:

print('{0:} ({1:.3f})'.format(vocablist[word], coef))

긍정 상위 50개 단어의 일부는 아래와 같습니다.

꿀잼 (3.283)

재밌었 (2.876)

재미있게 (2.793)

재미있었 (2.742)

재밌게 (2.732)

….

부정 상위 50개 단어의 일부는 아래와 같습니다.

최악 (-4.243)

노잼 (-3.437)

차라리 (-2.856)

아까워 (-2.850)

졸작 (-2.774)

…

### 하이퍼파라미터 튜닝

### 클래스 불균형 문제

## 나이브 베이즈 (Naïve Bayes)

### 나이브 베이즈의 작동 원리

이번에는 Naïve Bayes 알고리즘에 대해서 알아보도록 하겠습니다. 나이브 베이즈은 분류 문제에 사용이 되는 지도학습 알고리즘입니다.

어떠한 관측치의 종속변수가 취하는 값이 무엇인지 예측하기 위해, 각 관측치가 가지고 있는 독립변수 혹은 피쳐 정보를 이용해서 해당 관측치의 종속변수가 특정한 값을 취할 확률을 구하게 됩니다.

이를 구하기 위해 베이즈 공식을 사용합니다.

그리고 확률값이 제일 큰 값으로 종속변수의 값을 예측하게 됩니다.

예를 들어보겠습니다.

* + - i번째 관측치에 대해서, 인 경우
    - If , then 값은 0으로 예측, 그렇지 않으면 1로 예측
* 종속변수 (Y)가 취할 수 있는 값이 0, 1 두개인 경우, 각 값을 취할 확률은 다음과 같이 표현
  + - 는 데이터에 존재하는 Feature 들
    - 이는 종속변수의 값이 j일 확률이 Feature 들의 구체적인 값에 따라서 달라진다는 것을 의미
    - 베이즈 공식을 사용하면 다음과 같이 표현

확률이 높은 값으로 예측하게 됩니다.

두 확률의 대소 비교를 할 때 베이즈 공식 분모는 별로 중요한 역할을 하지 않습니다. 실제 분석에서도 분모의 값을 직접적으로 계산하지는 않습니다.

피처들이 특정한 값을 갖는 경우에 대해서 종속변수의 값이 1인 확률을 구해보도록 하겠습니다.

이 확률은 아래와 같이 됩니다.

우리가 구해야 하는 값은 입니다.

여기에서 왼쪽항을 먼저 살펴보겠습니다.

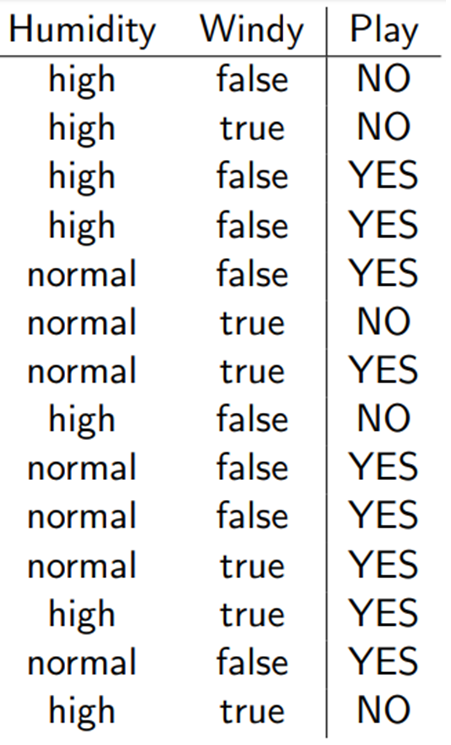
나이브 베이즈에서는 이를 구하기 위해서 각 피처들은 서로 독립이라는 가정을 사용합니다. 즉, 특정 피처가 어떠한 값을 취할 확률이 다른 피처에 의해서 영향을 받지 않는다는 것을 의미합니다.

이러한 경우, 다음과 같이 표현될 수 있습니다.

여기에서 는 학습데이터를 가지고 계산합니다.

그리고 도 학습데이터를 가지고 계산하게 됩니다.

예를 들어보겠습니다.



종속변수가 특정한 날에 골프를 쳤는지 그렇지 않은지에 대한 것이고, 이에 대한 피처 정보로 습도 정도 (humidity)와 바람 여부 (windy)를 존재한다고 가정합니다.

Humidity라고 하는 변수는 취할 수 있는 값이 high, normal, low 인데 지금은 high, normal 만 나와 있습니다. 그리고 windy라고 하는 변수는 false와 true의 값만을 취하게 됩니다. 이와 같은 학습데이터에 대해서, 새로운 관측치에 대한 종속변수를 예측하고자 한다라고 가정합니다. 그리고 해당 관측치의 피처 정보는 다음과 같습니다.

Humidity=normal, Windy=true

먼저, 이러한 feature 정보 하에서 종속변수가 Yes일 확률을 구해 보겠습니다.

여기에서 우리가 구해야 하는 것은 분자입니다. 먼저 왼쪽 항의 값부터 계산을 해보도록 하겠습니다.

humidity와 windy 피처가 서로 독립인 경우에는 다음과 같이 표현됩니다.

는 아래와 같이 계산됩니다.

종속변수의 값이 Yes인 날들 중에서 Humidity가 normal인 날들의 비중

이는 “종속변수의 값이 Yes인 날들 수 분에 종속변수의 값이 Yes인 날들 중에서 Humidity가 normal인 날들의 수”가 됩니다.

) 도 비슷하게 구할 수 있습니다.

그렇다면 P(Play=Yes)는 어떻게 구할까요? 이는 학습데이터에 존재하는 전체의 관측치들 중에서 종속변수의 값이 Yes인 관측치의 비중으로 구해집니다. 아래와 같습니다.

* + P(Play=Yes) = #Yes/(#Yes + #No) = 9/14

앞의 예에서는 독립변수가 모두 카테고리칼 변수였습니다. 이러한 경우에는 앞의 방법으로 확률을 하나의 독립변수가 특정한 값을 취할 확률을 쉽게 구할 수 있었습니다.

그렇다면 독립변수가 연속변수인 경우에는 그러한 확률을 어떻게 구할까요?

이러한 경우에는 두가지 방법을 사용할 수 있습니다. 하나는 연속변수를 카테고리칼 변수로 변환하는 방법입니다. 연속변수가 취하는 값을 특정한 기준에 따라서 몇개의 그룹으로 구분을 하는 것입니다.

또 다른 방법은 확률을 구하기 위해서 특정 연속확률분포의 pdf를 사용하는 것입니다. 예를 들어 정규분포를 사용할 수 있는 것입니다.

독립변수들이 count variable 인 경우, 즉 변수의 값들이 특정 사건이 발생한 횟수를 의미하는 경우, 우리는 해당 독립변수들에 대한 데이터가 multinomial 분포 즉, 다항분포를 따른다고 합니다. 이러한 경우에는 multinomial NB를 사용하게 됩니다. 대표적인 예가 텍스트 데이터가 됩니다. 감성분석을 예로 들어보겠습니다. 특정 문서에서 어떠한 단어들이 몇번 사용되었는지에 따라서 해당 문서의 종속변수를 긍정과 부정으로 예측하는 것을 감성분석이라고 합니다. 이러한 경우, 우리가 사용하는 feature 정보는 무엇인가요?

단어들이 됩니다. 그리고 단어 feature가 갖는 값은 각 문서에서 해당 단어가 몇번 사용되었는지에 대한 frequency 정보가 됩니다. 즉 count 변수라고 생각할 수 있습니다. 하나의 문서에 대한 이러한 단어들에 대한 데이터는 다항 분포를 갖는다고 할 수 있습니다. 따라서 감성분석을 할 때 사용할 수 있는 NB가 multinomial NB가 됩니다.

multinomial NB를 잘 이해하기 위해서는 다항 분포에 대해서 이해하는게 필요합니다.

다항 분포는 이항분포의 일반화된 분포라고 생각할 수 있습니다. 그렇다면 이항분포는 무엇일까요?

이항 분포 (Binomial distribution)

‘성공’과 ‘실패’ 둘 중 하나의 값이 결과로 나올 수 있는 시행(이러한 시행을 베르누이 시행이라고합니다)을 n 번 수행할 때, 나오는 ’성공’의 횟수를 값으로 취하는 변수를 X라고 하는 경우, 이 X 변수는 이항분포를 따릅니다. 그리고 보통 ‘성공’을 1로, ‘실패’를 0으로 표현합니다.

이항 분포의 확률밀도함수(probability mass function, PMF)는 아래와 같습니다.

위 식에서 은 베르누이 시행의 횟수를 의미하고, 는 한 번 시행했을 때 1의 값이 나올 확률을 의미합니다. 는 n번의 시행에서 1의 값이 나온 시행의 횟수가 됩니다. 는 조합(combination)을 의미하며, 아래와 같이 계산됩니다.

이항 분포를 갖는 대표적인 예가, 동전을 n번 던졌을 때 나오는 앞면의 수를 취하는 변수가 됩니다. 여기에서 동전을 한번 던졌을 때 나올 수 있는 값은 앞면과 뒷면 두개가 되고, 앞면을 1 또는 ‘성공’으로 뒷면을 0 또는 ‘실패’라고 간주할 수 있습니다. 그리고 한 번 던졌을 때 앞면이 나올 확률이 p가 되는 것입니다.

다항 분포 (Multinomial distribution)

다항 분포는 이항 분포의 일반화된 분포입니다. 이항 분포의 경우, 한 번의 시행에서 나올 수 있는 값이 오직 두 개 (즉, ‘성공’과 ‘실패’ 혹은 1과 0)인 반면, 다항 분포의 경우는 한 번의 시행에서 나올 수 있는 값이 여러 개(즉, 세 개 이상)가 됩니다. 주사위를 던지는 경우가 그러한 예입니다. 주사위를 한 번 던지는 경우 나올 수 있는 값의 수는 여섯 개가 됩니다. 그리고 각 값이 나올 확률은 (주사위가 fair하다고 가정한다면) 1/6이 됩니다. 여러 개의 결과값을 가질 수 있는 시행을 n 번 수행했을 때 각 결과값이 나오는 횟수를 값으로 취하는 변수들의 분포를 다항 분포라고 합니다. 주사위를 던지는 경우의 예에서는 변수가 여섯 개가 존재하는 것입니다. 이를 라고 표현하겠습니다. 주사위를 n번 던졌을 때 1의 눈이 나오는 횟수를 결과로 취하는 변수가 , 2의 눈이 나오는 횟수를 결과로 취하는 변수가 가 되는 것입니다. n번 시행에 대해서, 각 변수가 취하는 값을 라고 하는 경우에 다음과 같은 방정식이 만족하게 됩니다.

한 번 시행을 했을 때 K개의 결과가 나올 수 있고, 결과가 나올 확률을 라고 하는 경우에 다음 방정식을 만족해야 합니다.

주사위 예의 경우는 K의 값이 6이 되고, 아래 방정식이 만족해야 합니다.

주사위의 경우, 은 주사위를 한 번 던졌을 때 1의 눈이 나올 확률 (즉, )을 의미합니다. 주사위의 경우 모든 이 됩니다.

한 번 시행을 했을 때 K개의 결과가 나올 수 있는 시행을 n번 수행한 경우에 대한 다항 분포의 확률질량함수는 아래와 같이 정의됩니다.

위 식에서 는 시행을 n 번 수행했을 때 1의 값이 나오는 횟수를 값으로 하는 변수를 의미합니다. 는 해당 시행을 한 번 수행했을 때 결과가 나올 확률을 나타냅니다.

참고로 다항 분포에서 시행의 횟수가 1인 경우 (즉, n=1인 경우)의 분포를 카테고리 (Categorical) 분포라고 합니다.

### 다항 나이브 베이즈를 이용한 감성 분석

예를 들어보겠습니다.

문서 D = ‘I like the movie and you also like it’

이라는 영화평의 긍부정을 예측하기 위해 다항 나이브 베이즈 알고리즘을 사용한다고 가정합니다. 이를 위해서 우리가 비교해야 하는 값은

P(Positive|D)와 P(Negative|D) 입니다.

P(Positive|D)를 먼저 살펴보겠습니다. 여기서 D는 예측을 하고자 하는 문서에 대한 데이터로 이는 다음과 같이 표현될 수 있습니다.

P(Positive|also=1, and=1, i=1, it=1, like=2, movie=1, the=1, you=1)

원래는 텍스테 데이터에 존재하는 단어들 중에서 문서 D에서 사용되지 않은 단어들에 대한 정보도 포함되어야 하지만, 그러한 단어들은 위 확률에 영향을 주지 않으므로 생략합니다.

그리고 also, and, the 와 같은 단어들은 별의미가 없는 불용어로 간주되어 일반적으로 전처리 과정에서 제거되지만 여기서는 설명을 위해 포함하도록 합니다.

P(Positive|also=1, and=1, i=1, it=1, like=2, movie=1, the=1, you=1) 는 베이즈 규칙에 따라 다음과 같이 표현됩니다.

P(Positive|also=1, and=1, i=1, it=1, like=2, movie=1, the=1, you=1)

그리고 P(Negative|also=1, and=1, i=1, it=1, like=2, movie=1, the=1, you=1)

둘의 값을 비교하기 위해서 분자의 값만 계산하면 됩니다. 이를 위해서 학습 데이터가 사용됩니다.

는 학습 데이터에 존재하는 전체 문서들 중에서 긍정 문서들의 비중이 됩니다.

는 다항 분포를 이용해서 계산됩니다.

이 다항 분포의 경우, 한 번의 시행에서 나올 수 있는 결과의 수는 Vocabulary에 있는 단어의 수가 됩니다. 그리고 n은 문서에서 사용된 전체 단어의 수가 됩니다. 위의 예에서는 n=9입니다.

다항 분포의 확률질량함수는 아래와 같습니다.

여기서 는 vocabulary에 있는 단어를 의미합니다. 문서 D에 대해서 의 값을 위의 확률질량함수를 이용해서 계산해야 하는 것입니다.

가 됩니다. 즉, 긍정의 문서에서 단어 가 출현할 (혹은 사용될) 확률을 의미합니다. 이는 다음과 같이 계산됩니다.

여기에서 는 학습 데이터에 존재하는 긍정의 문서들에서 단어 가 사용된 횟수를 의미하고, 는 긍정인 문서들에서 전체 단어들이 사용된 횟수가 됩니다. 은 텍스트 데이터에 존재하는 (고유한) 단어의 수를 의미합니다. 는 분자가 0이 되는 것을 방지하기 위해서, 그리고 확률을 계산할 때 사용된 문서들에서 사용되지 않은 단어들을 고려하기 위해 사용된 하이퍼파라미터 (스무딩 항(smoothing term)이라고 합니다)로, 인 경우는 라플라스 스무딩 (Laplace smoothing)이라고 하고, 인 경우는 리드스톤 스무딩 (Lidstone smoothing)이라고 합니다.

위 식에서 과 은 클래스 (즉, Positive 또는 Negative)에 영향을 받지 않으므로 결과에 영향을 미치지 않아 계산에서 생략해도 상관없습니다.[[45]](#footnote-46)

### 파이썬 코딩

파이썬 코드는 MNB\_sentiment.ipynb 파일을 참고하세요.

## 결정 트리와 앙상블 방법

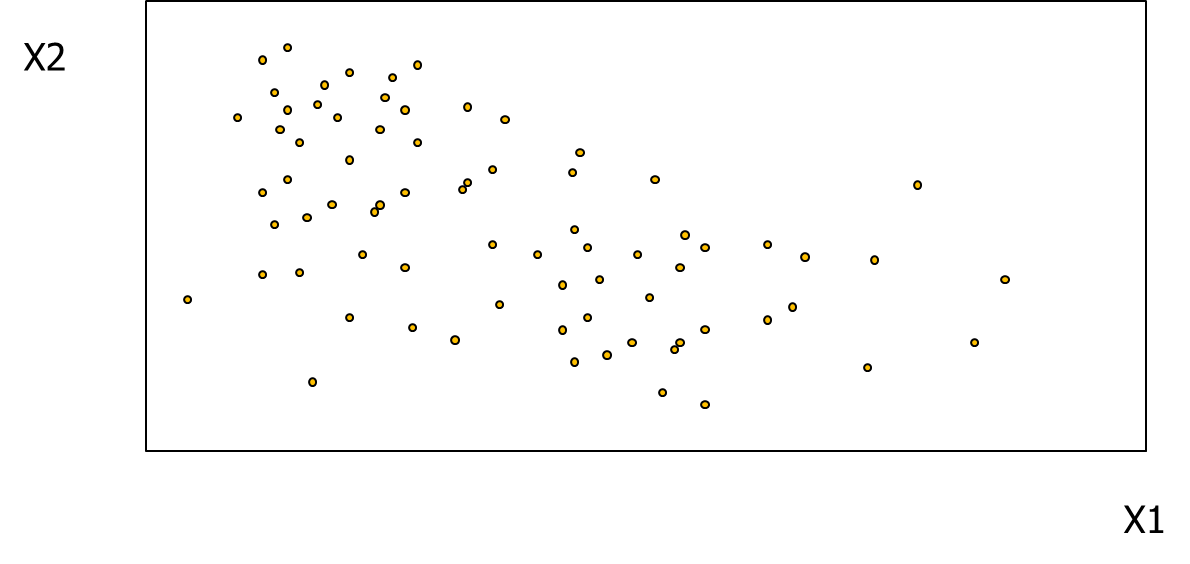
특성 정보 (feature)가 너무 많지 않은 정형 데이터에 대해서 적용되어 좋은 성능을 내는 지도학습 알고리즘으로 Random Forest, XGBoost, Light GBM 등이 있습니다. 이러한 알고리즘은 앙상블 (Ensemble) 기반의 알고리즘으로 특정 모형을 하나만 사용하는 것이 아니라 여러 개를 사용하여 모형의 성능을 높이는 방법입니다 (앙상블 방법에 대해서는 잠시 후에 자세히 설명합니다). 이러한 앙상블 기반의 방법들은 일반적으로 기본 모형으로 결정 트리 (Decision Tree) 모형을 사용합니다. 따라서, 본 장에서는 결정 트리에 대해 먼저 설명하고 이후 여러 가지의 앙상블 기반 알고리즘에 대해서 설명하도록 하겠습니다.

## 결정 트리 (Decision Tree)

결정 트리는 데이터에 있는 관측치들을 특성 정보 (features, 또는 독립변수)의 값에 따라서 종속변수의 값이 유사한 여러 개의 그룹으로 분리하고, 각 그룹에 속한 관측치들의 종속변수 값을 동일한 값으로 예측하는 알고리즘입니다. 데이터를 여러 개의 그룹으로 분리하는 과정이 나무와 닮았다고 해서 결정 트리라는 이름을 갖습니다.

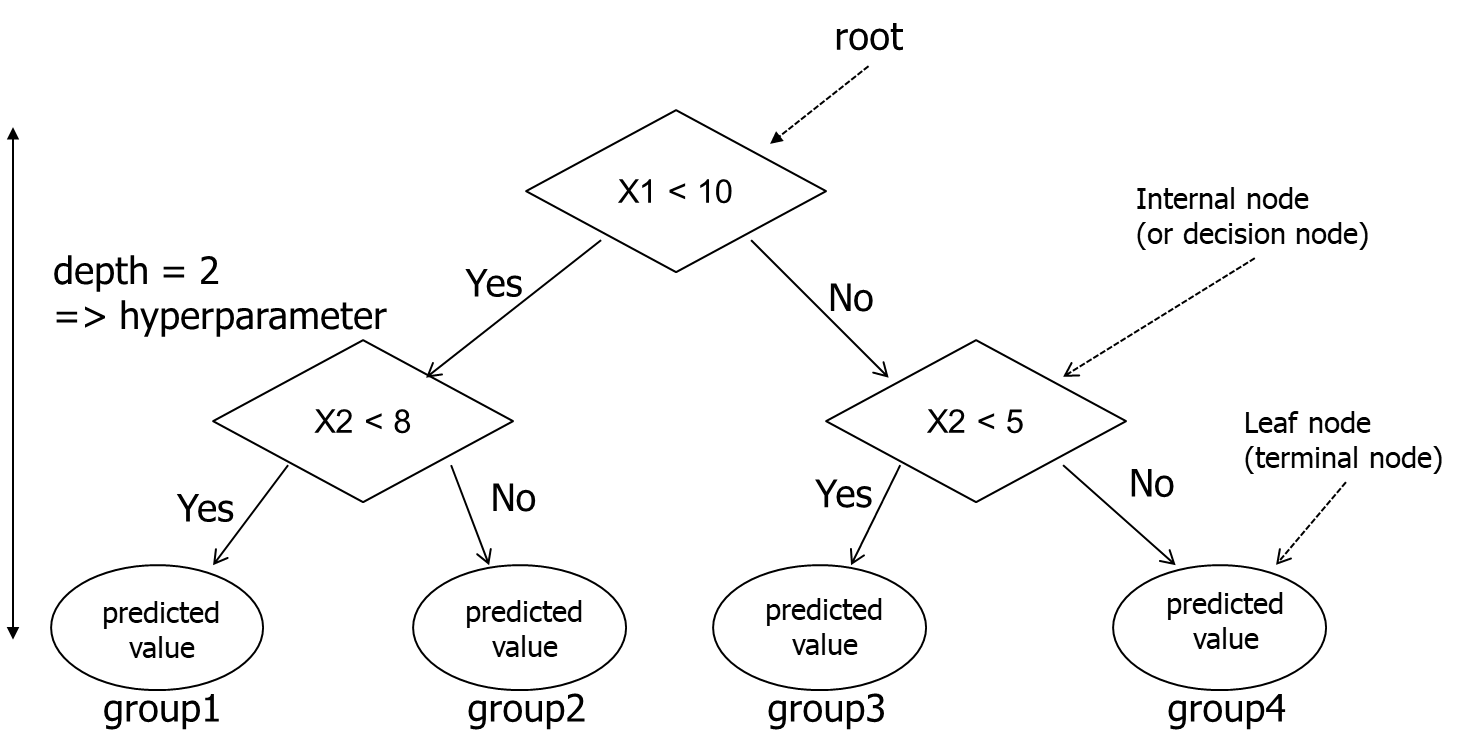
결정 트리는 회귀문제와 분류문제에 모두 적용될 수 있습니다. 회귀 문제에 적용되는 트리를 회귀 트리 (regression tree)라고 하고, 분류 문제에 적용되는 트리를 분류 트리 (classification tree)라고 합니다.

결정 트리의 작동 방식을 설명하기 위해 다음과 같은 데이터가 있다고 가정합니다. 해당 데이터에 존재하는 각 관측치는 두 개의 특성 정보 (X1, X2)와 종속변수를 갖습니다. 각 관측치는 두 개 특성 정보를 이용해서 벡터로 표현됩니다. 따라서 다음과 같이 2차원 공간의 점(point)으로 표현됩니다.



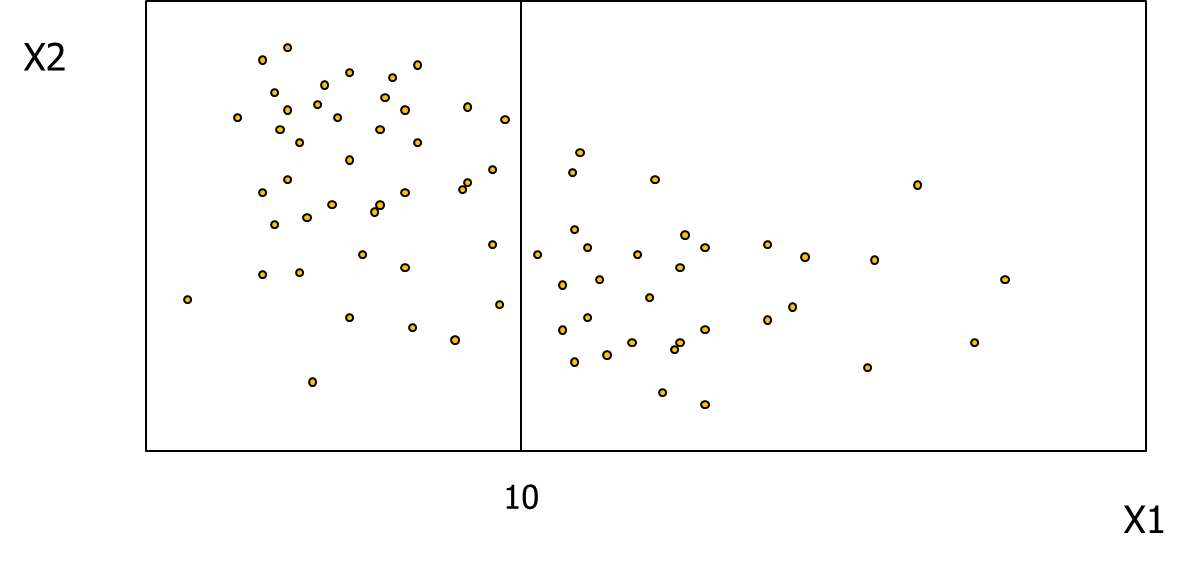
**그림 10.1 예제 데이터**

결정 트리는 위와 같이 존재하는 관측치들을 해당 특성 정보, 즉 X1과 X2의 값을 이용해서 분리하게 됩니다. 다음과 같은 트리를 이용해서 데이터가 구분이 된다고 가정을 해보겠습니다.



위 그림에서 마름모에 해당하는 노드를 결정 노드 (decision node, 또는 internal node)라고 합니다. 결정 노드에서는 특정 특성 정보의 특정한 값을 기준으로 데이터를 두 개의 그룹으로 구분합니다. 제일 위에 존재하는 결정 노드를 루트 노드라고 합니다. 분리 과정의 결과로 도출된 제일 아래 있는 타원을 리프 노드 (leaf node 또는 terminal node라고도 합니다)라고 합니다. 리프 노드에는 종속변수의 값이 상대적으로 유사한 관측치들이 포함되어 있습니다 (리프 노드는 분리 과정을 통해 도출된 종속변수의 값들이 유사한 관측치들이 포함된 하나의 그룹으로 간주될 수 있습니다). 그리고 루트 노드에서 리프 노드까지의 길이를 트리의 깊이 (depth)라고 합니다. 위 트리의 깊이는 2입니다.

위의 트리에 따르면 관측치들은 가장 먼저 X1을 기준으로 서로 다른 두 개의 그룹으로 분리됩니다. 구분의 기준이 되는 값은 10입니다 (그룹 나누기를 할 때 어떠한 변수와 해당 변수의 어떠한 값을 사용하는지에 대해서는 나중에 다시 설명하겠습니다). 즉, 전체 관측치들이 X1이 10보다 작은 그룹과 그렇지 않은 그룹으로 구분이 되는 것입니다 (아래 그림 참고).



그 다음에는 첫 번째 과정에서 구분이 된 **두 개의 그룹 각각에 대해서** 다시 한 번 분리의 작업을 하게 됩니다. 즉, X1이 10보다 작은 왼쪽 그룹을 특정한 변수의 특정한 값을 이용해서 다시 한 번 두 개의 그룹으로 분리를 하고, X1이 10 이상인 그룹에 대해서도 동일한 작업을 수행합니다. 위의 트리에 따르면, X1이 10보다 작은 그룹은 X2의 값 8을 기준으로 다시 한 번 두 개의 그룹으로 분리가 되었고, X1이 10 이상인 그룹은 X2의 값 5를 기준으로 서로 다른 두 개의 그룹으로 분리가 되었습니다. 이러한 분리의 결과로 우리는 아래 같은 서로 다른 네 개의 그룹을 얻을 수가 있습니다. 각 그룹이 하나의 리프 노드가 되는 것입니다.



결정 트리는 이러한 과정을 특정한 기준이 만족될 때 가지 반복합니다. 예를 들어, 특정 깊이 (depth)까지 데이터를 분할하거나, 분할시 발생하는 하나의 그룹 안에 존재하는 관측치의 수가 특정 수 보다 클 때 까지 반복합니다. 분할시 사용되는 기준에 대해서는 잠시 후 조금 더 자세히 설명하겠습니다.

결정 트리는 독립변수 값을 이용해서 데이터에 존재하는 관측치들을 서로 다른 그룹으로 분리하는 과정을 여러 번 수행한 후, 같은 그룹 (즉, 리프 노드)에 속한 관측치들의 종속 변수 값을 동일한 값으로 예측을 합니다. 예측을 하는 방식은 우리가 풀고자 하는 문제가 회귀 문제인지 아니면 분류 문제인지에 따라서 달라집니다. 즉, 종속변수가 연속변수인지 이산변수(명목변수)인지에 따라서 달라집니다. 회귀 문제인 경우에는 같은 그룹에 속해 있는 관측치들이 갖는 종속변수 값들의 평균값을 종속변수의 예측치로 사용하고, 분류 문제인 경우에는 종속변수가 취할 수 있는 값들 중에서 최빈값 (mode)을 종속변수의 예측치로 사용합니다. 즉, 해당 그룹에 존재하는 관측치들 중에서 제일 많은 관측치가 갖는 종속변수의 값을 예측치로 사용하는 것입니다.

회귀 문제와 분류 문제에서 각 그룹의 종속변수 값이 어떻게 예측되는지에 대한 구체적인 예를 살펴 보겠습니다.

① 회귀 문제

회귀 문제의 경우는, 특정 그룹에 속한 데이터 포인트들이 갖는 종속변수 값의 평균값으로 예측을 합니다. 에를 들어, 특정 그룹에 열 개의 관측치가 속해 있고, 각 관측치가 다음과 같은 연속변수 형태의 종속변수 값을 갖는다라고 가정하겠습니다.

(2, 3, 4, 4, 3, 6, 4, 10, 2, 12)

이러한 경우에 해당 관측치들의 종속변수 값은 위 종속변수 값들의 평균인 5 (= (2+ 3+ 4+ 4+ 3+ 6+ 4+ 10+ 2+ 12)/10)로 예측이 됩니다. 이는 아래와 같이 표현될 수 있습니다.

, where = 그룹 k에 속한 관측치의 수

② 분류 문제

분류 문제의 경우는 특정 그룹에 속한 데이터 포인트들의 종속변수 값의 최빈값으로 예측을 합니다. 이번에는 특정 그룹에 열 개의 관측치가 속해 있고, 각 관측치가 다음과 같은 명목변수 형태의 종속변수 값을 갖는다라고 가정합니다.

(2, 0, 1, 1, 1, 2, 1, 1, 0, 1)

이러한 경우에는 가장 많은 관측치들이 갖는 종속변수의 값 (즉, 최빈값)은 1이 되므로, 해당 그룹에 속해 있는 관측치들의 종속변수는 1로 예측이 됩니다.

## 참고 시작 ##

참고

# of branches

sklearn에서는 binary split만 제공

<https://datascience.stackexchange.com/questions/51983/is-it-possible-to-output-more-than-2-nodes-away-from-a-node-in-a-decision-tree>

## 참고 끝 ##

**결정 노드에서 데이터 분할 시 사용되는 변수와 변수의 값**

그렇다면 특정 결정 노드에서 어떠한 변수의 어떠한 값 (cutpoint)으로 관측치들을 서로 다른 두개의 그룹으로 분할을 할까요? 이 때는 두 개의 그룹으로 구분이 되었을 때 발생하는 에러의 정도를 최소화하는 방법으로 두 개의 그룹으로 분할하게 됩니다. 특정한 그룹에 존재하는 에러 정도를 측정하는 방법 역시 회귀 문제인지 혹은 분류 문제인지에 따라서 달라집니다.

① 회귀 문제에서의 에러 정도

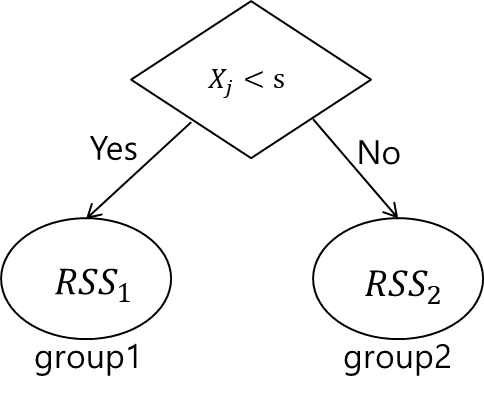
회귀 문제에서는 분할시 발생하는 각 그룹에 대한 에러 정도를 측정하기 위해 잔차 제곱의 합 (residual sum of squares, RSS)을 사용합니다. 하나의 그룹에 대한 RSS는 다음과 같이 계산됩니다. 아래는 그룹 에 대한 RSS를 나타냅니다.

는 그룹 에 속하는 관측치 의 실제 종속변수 값을 의미하며, 는 그룹 에 대한 종속변수의 예측치입니다. 해당 그룹에 속하는 관측치의 실제 종속변수 값에서 해당 그룹에 대한 예측치의 차이의 제곱으로 계산하게 되는 것입니다.

특정한 결정 노드에서 분할의 결과로 서로 다른 두 개의 그룹이 생기게 될 텐데, 각 그룹에 대한RSS 값의 합을 최소화 하는 변수 와 해당 변수의 값 를 결정하게 됩니다. 아래와 같이 표현될 수 있습니다.

는 분할시 발생하는 두 개의 그룹 중 첫 번째 그룹에 대한 RSS가 되고, 아래와 같이 표현될 수 있습니다.

이를 그림으로 표현하면 아래와 같습니다.



② 분류 문제에서의 에러 정도

분류 문제에서의 에러 정도를 측정하기 위해서는 일반적으로 지니 인덱스 (Gini index) 또는 엔트로피(entropy) 값을 사용합니다. 즉, 두 개의 그룹으로 분할 되었을 때 존재하는 각 그룹별 지니 인덱스값의 합이나 엔트로피 값의 합[[46]](#footnote-48)을 최소화하는 변수와 해당 변수의 값을 이용해서 두 개의 서로 다른 그룹으로 분할합니다.

종속변수가 취하는 값이 {1, 2, …, K}인 경우, 특정 그룹에 대한 지니 인덱스는 다음과 같이 계산이 됩니다.

는 그룹 에 대한 지니 인덱스를 의미합니다. 는 그룹 에서 종속변수의 값이인 관측치의 비중을 의미하며, 이는 다음과 같이 계산됩니다.

는 그룹 에 속한 모든 관측치의 수이며, 는 해당 그룹에 속한 관측치들 중에서 종속변수의 값이 인 관측치의 수를 의미합니다.

그렇다면 특정 그룹의 지니 인덱스 값은 언제 작아지게 될까요? 지니 인덱스의 값은 해당 그룹에 속해 있는 모든 관측치가 동일한 값을 가질 때 가장 작아지게 됩니다. 설명을 위해, 열 개의 관측치로 구성된 그룹이 하나 있다고 가정하겠습니다. 종속변수가 취할 수 있는 값이 0과 1인 경우에 대해서 아래와 같이 서로 다른 세 가지 케이스에 대해서 지니 인덱스를 계산해 보겠습니다.

Case 1: (0,0,0,0,0,0,0,0,0,0) (모든 관측치의 종속변수 값이 0인 경우)

Case 2: (0,0,0,0,0,1,1,1,1,1)

Case 3: (0,0,0, 1,1,1,1,1,1,1)

Case 1에 대한 지니 인덱스를 먼저 계산해 보겠습니다. 종속변수가 취하는 값이 0과 1 두 개이기때문에 아래와 같이 표현됩니다.

여기서 는 해당 그룹에 속한 관측치들 중에서 종속변수의 값이 0인 관측치의 비중을, 는 종속변수의 값이 1인 관측치의 비중을 의미합니다. 해당 그룹에 속한 전체 관측치의 수가 10이고, 모든 관측치의 종속변수 값이 0이기 때문에, 과 은 아래와 같이 계산됩니다.

따라서 해당 그룹의 지니 인덱스 값은 아래와 같습니다.

Case 2의 경우는 0의 값을 갖는 관측치의 수와 1의 값을 갖는 관측치의 수가 각각 5이므로 지니 인덱스의 값은 아래와 같습니다.

Case 3의 경우의 지니 인덱스 값은 아래와 같습니다.

위에서 볼 수 있는 것 처럼 지니 인덱스의 값은 그룹에 속한 모든 관측치가 동일한 종속변수의 값을 갖는 경우 0으로 제일 작아지고, 서로 다른 값을 갖는 관측치의 수가 동일한 경우 (Case 2) 그 값이 제일 커집니다. 분류 문제의 경우는 종속변수의 값이 최빈값으로 예측되므로 Case 1의 경우는 종속변수의 값이 0으로 예측됩니다. 즉, 모든 관측치의 종속변수 값이 정확히 예측되게 됩니다. 하지만, Case 2의 경우는 종속변수의 값이 0 또는 1로 예측됩니다. 즉, 그룹에 속한 관측치들 중 반에 해당하는 관측치들의 종속변수 값이 틀리게 예측되는 것입니다.

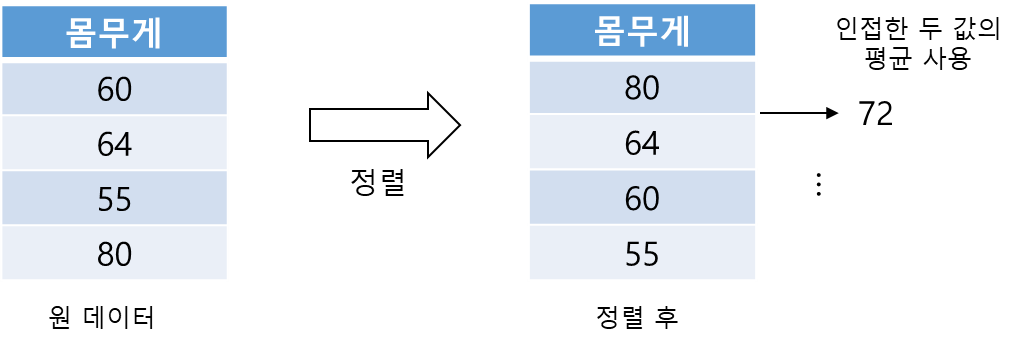
이번에는 엔트로피에 대해서 살펴보겠습니다. 종속변수가 취하는 값이 {1, 2, …, K}인 경우, 하나의 그룹에 대한 엔트로피(Entropy)[[47]](#footnote-49)의 값은 다음과 같이 계산이 됩니다.

는 그룹 에 대한 엔트로피를 의미합니다. 지니 인덱스에서와 마찬가지로 는 그룹 에서 종속변수의 값이인 관측치의 비중을 의미합니다. 엔트로피도 지니 인덱스와 마찬가지로 그룹에 속한 모든 관측치들의 종속변수 값이 동일한 경우 그 값이 가장 작아집니다.

지니 인덱스와 엔트로피의 경우, 모형의 성능 관련해서는 큰 차이가 없습니다. 엔트로피의 경우 로그 계산을 해야하기 때문에 데이터의 양이 많은 경우 지니 인덱스 보다 계산 속도가 조금 느릴 수 있습니다. sklearn 모듈에서는 기본적으로 지니 인덱스를 사용합니다.

그렇다면, 결정 노드에서는 각 변수에 대해서 어떠한 값을 이용해서 분할 시도를 하게 될까요?

데이터에 존재하는 변수의 실제값을 활용합니다. 구체적으로 어떠한 값을 사용하는지는 변수가 연속 변수인지 혹은 범주형 변수인지에 따라서 달라집니다. 범주형 변수 경우는 변수가 갖고 있는 실제값을 그대로 사용합니다. 하지만 연속 변수는 실제의 값을 그대로 사용하지 않고 인접한 값들의 평균값을 사용합니다. 설명을 위해서, 연속 변수의 경우 어떠한 값들이 사용되는지를 설명하기 위해 구체적인 예를 살펴보겠습니다. 몸무게라는 연속 변수가 있고 해당 변수가 갖는 실제 값이 그림XX의 왼쪽과 같다라고 가정합니다.



결정 트리는 일단 해당 변수의 값들을 크기순으로 정렬한 뒤 인접한 두 값의 평균값을 사용해서 결정 노드에서 데이터를 서로 다른 두 개의 그룹으로 분할을 시도합니다.

그렇다면 분할은 언제까지 계속 진행되는 것일까요?

중단 기준 (stopping criterion)이 만족될때 까지 계속해서 분할을 합니다. 중단 기준으로 사용되는 주요한 하이퍼파라미터에는 max\_depth, min\_samples\_split, min\_samples\_leaf, max\_leaf\_nodes 등이 있습니다. 각 하이퍼파라미터에 대해서 알아보겠습니다.

max\_depth

max\_depth는 트리가 가질 수 있는 최대 깊이를 의미합니다. 예를 들어, max\_depth=2라고 설정하면 해당 트리의 최대 깊이는 2가 됩니다. 더 분할할 수 있다고 할지라도 트리의 깊이가 2가 넘게는 분할하지 않습니다. max\_depth의 값이 커질수록 트리의 형태가 더 복잡해 집니다. 즉, 과적합이 발생할 가능성이 커지게 되는 것입니다.

min\_samples\_split

min\_samples\_split는 하나의 노드가 추가적으로 분할되기 위해서 가져야 하는 최소한의 관측치 수를 나타냅니다. 예를 들어, min\_samples\_split=10인 경우 포함된 관측치의 수가 10 미만인 리프 노드는 더 이상 분할 되지 않습니다. 이 값이 작을수록 분할이 더 많이 진행되기 때문에 트리가 더 복잡해 지고 따라서 과적합 문제가 발생할 가능성이 증가합니다.

min\_samples\_leaf

min\_samples\_leaf는 리프 노드가 가져야 하는 최소한의 관측치 수를 의미합니다. 분할 후 리프 노드에 존재하는 관측치의 수가 이 값보다 작게 되면 분할이 진행되지 않습니다. 예를 들어, min\_samples\_leaf=10인 경우, 분할을 했는데 리프 노드에 속한 관측치의 수가 10 미만인 경우에는 해당 분할은 진행되지 않습니다. 이 값이 작을수록 트리가 복잡해 집니다.

max\_leaf\_nodes

max\_leaf\_nodes는 트리가 가질 수 있는 최대 리프 노드의 수를 의미합니다. 분할을 했을 때 존재하는 전체 리프 노드의 수가 이 값보다 커지는 경우 해당 분할은 진행되지 않습니다. 이 값이 커질수록 트리가 복잡해 집니다.

결정 트리의 경우 하이퍼파라미터와 각 하이퍼파라미터가 취할 수 있는 값의 수가 많기 때문에 하이퍼파라미터 튜닝을 위해서 일반적으로 Grid Search 방법 등을 사용합니다.

Pros vs Cons of Decision Trees

Advantages:

The main advantage of decision trees is how easy they are to interpret. While other machine Learning models are close to black boxes, decision trees provide a graphical and intuitive way to understand what our algorithm does.

Compared to other Machine Learning algorithms Decision Trees require less data to train.

They can be used for Classification and Regression.

They are simple.

They are tolerant to missing values.

Disadvantages

They are quite prone to over fitting to the training data and can be sensible to outliers.

They are weak learners: a single decision tree normally does not make great predictions, so multiple trees are often combined to make ‘forests’ to give birth to stronger ensemble models. This will be discussed in a further post.

### 파이썬 코딩

#### 예제 데이터: 아이리스 꽃 데이터[[48]](#footnote-50)

결정 트리가 작동하는 방식을 보다 잘 이해하기 위해서 간단한 예제 데이터에 결정 트리를 적용해 보겠습니다.[[49]](#footnote-51)

기본적으로 다음과 같이 numpy 모듈과 pandas 모듈을 임포트 합니다.

import pandas as pd

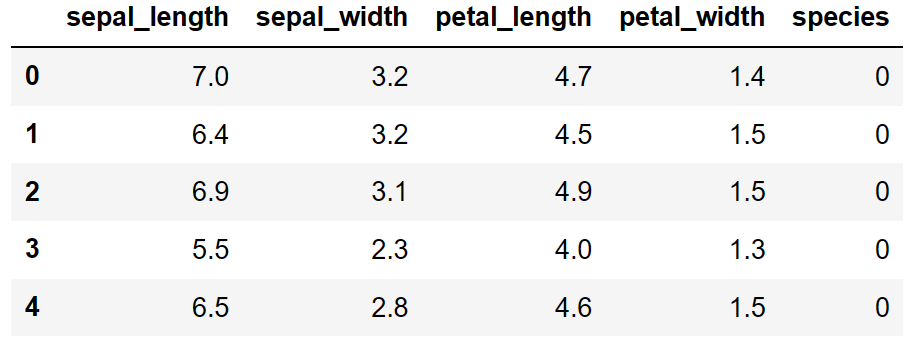
import numpy as np

pandas의 read\_csv() 함수를 이용해서 데이터 프레임 형태로 데이터를 불러 옵니다.

df\_iris = pd.read\_csv('iris\_data.csv')

해당 데이터는 아래와 같습니다.

df\_iris.head()



특성 정보가 4개 (꽃받침의 길이와 너비, 꽃잎의 길이와 너비)이고, 종속변수는 species (꽃의 종)이됩니다. 종속 변수가 취할 수 있는 값은 0과 1이고, 0은 versicolor를 1은 virginica를 의미합니다. 각 종은 아래 그림과 같이 생겼습니다.



종속변수와 특성 정보를 아래와 같이 구분하여 저장합니다.

X = df\_iris.iloc[:,:-1] # 모든 행 포함, 마지막 열은 제외한다는 것을 의미합니다.

y = df\_iris.iloc[:,-1] # 모든 행 포함, 마지막 열만 포함한다는 것을 의미합니다.

학습 데이터와 평가 데이터로 구분합니다.

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=.2)

다음과 같이 분류 결정 트리를 사용하기 위해 sklearn에서 제공되는 DecisionTreeClassifier 클래스를 임포트합니다.

from sklearn.tree import DecisionTreeClassifier

일단 DecisionTreeClassifier() 생성자 함수가 갖는 파라미터 중에서 max\_depth의 값을 2로 설정해서 모형을 만들어 보도록 하겠습니다. 이렇게 하면 학습을 통해 생성되는 트리의 최대 깊이가 2가 됩니다.

model = DecisionTreeClassifier(max\_depth=2)

fit() 함수를 이용해서 학습을 하고, predict()을 이용해서 평가 데이터에 존재하는 관측치들의 종속변수 값을 예측합니다.

model.fit(X\_train, y\_train)

y\_preds = model.predict(X\_test)

classification\_report() 함수를 사용해서 평가 데이터에 대한 모형의 성능 지표를 확인합니다.

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_preds))

precision recall f1-score support

0 1.00 0.80 0.89 10

1 0.83 1.00 0.91 10

accuracy 0.90 20

macro avg 0.92 0.90 0.90 20

weighted avg 0.92 0.90 0.90 20

가중 평균 f1 score의 값이 0.9가 나오는 것을 확인할 수 있습니다.

학습의 결과로 도출된 결정 트리 모형을 시각화 해 보겠습니다. 시각화를 위해서 아래와 같이 sklearn에서 제공되는 tree 모듈의 plot\_tree() 함수를 사용합니다. plot\_tree() 함수의 인자로는 학습을 마친 결정 트리 모형 (즉, model 객체)을 입력합니다.

from sklearn import tree

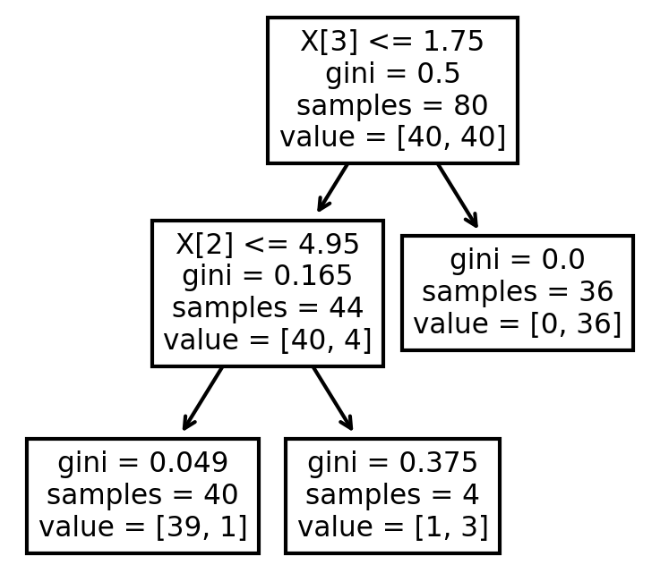
import matplotlib.pyplot as plt

fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (3,3), dpi=300)

tree.plot\_tree(model) # 모형의 시각화

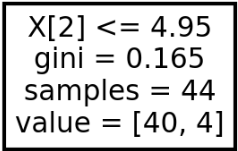
plt.show() # 화면에 출력 결정 트리를 이용한 감성 분석

시각화 결과는 아래와 같습니다.



첫 번째 결정 노드 (즉, 루트 노드)의 경우 X[3]의 특성 정보의 1.75 값을 기준으로 서로 다른 두 개의 그룹으로 분할된 것을 알 수 있습니다. X에는 네 개의 열이 있기 때문에 X[3]은 네 번째 열에 해당하는 특성 정보 (즉, petal\_width, 꽃잎의 너비)를 의미합니다. 즉, 꽃잎 너비의 값 1.75를 기준으로 학습 데이터에 있는 관측치가 두 개의 그룹으로 구분된 것입니다. 첫 번째 결정 노드에 존재하는 전체 관측치의 수는 80 (즉, samples = 80)으로 이는 학습 데이터에 존재하는 전체 관측치의 수와 동일합니다. 그리고 value = [40, 40]은 80개의 관측치들 중에서 종속변수의 값이 0인 관측치가 40개, 그리고 1의 값을 갖는 관측치의 수가 40이라는 것을 의미합니다. 따라서 해당 노드에서의 지니 인덱스 값은 0.5가 됩니다.

X[3] <= 1.75를 만족하는 관측치들은 왼쪽 노드로 그렇지 않은 관측치들은 오른쪽 노드로 분할됩니다. 레벨 1의 첫 번째 노드 (즉, 그림 XX)를 보면 해당 노드에 속한 관측치의 수는 44인 것을 알 수 있습니다. 즉, 루트 노드에서의 분할 기준이었던 X[3] <= 1.75을 만족하는 관측치가 학습 데이터에 44개 존재한다는 것을 의미합니다. 그 중 40개의 관측치가 0의 값을 갖고, 4개의 관측치가 1의 값을 갖습니다. 그에 해당하는 지니 인덱스는 0.165가 됩니다. 해당 노드에서 사용된 분류 기준은 X[2] <= 4.95입니다. X[2]는 X에 저장되어 있는 세 번째 열의 특성 정보 (즉, petal\_length, 꽃잎의 길이)를 의미합니다.



나머지 노드들의 정보도 비슷한 식으로 해석될 수 있습니다.

그림 XX에서 볼 수 있듯이 결과로 나온 트리의 깊이는 2입니다. 이는 DecisionTreeClassifier() 생성자 함수가 갖는 max\_depth의 값을 2로 설정했기 때문에 그렇습니다.

로지스틱 회귀 모형의 경우, 각 특성 정보 (즉, 독립변수)가 종속변수를 설명하는데 있어서 얼마나 중요한 역할을 하는지를 학습의 결과로 도출된 모형의 파라미터 절대값을 이용해서 파악할 수 있었습니다 (즉, 모형 파라미터의 절대값이 클수록 종속변수에 대한 설명력이 큰 것을 의미합니다).

결정 트리의 경우 각 특성 정보의 중요도를 파악하기 위해서 DecisionTreeClassifier 클래스가 갖고 있는 feature\_importances\_ 변수에 저장되어 있는 값을 확인하면 됩니다. feature\_importances\_ 변수는 각 특성 정보들이 종속변수를 설명하는데 있어서 얼마나 중요한 역할을 하는지를 나타냅니다. 위 모형에 대해 feature\_importances\_ 변수값을 확인해 보겠습니다 (아래 코드 참고).

model.feature\_importances\_

array([0. , 0. , 0.10458898, 0.89541102])

보다 해석하기 쉽게 아래 코드를 사용해서 결과를 출력해 보겠습니다.

feature\_dict = dict(zip(X.columns, model.feature\_importances\_))

[(k,v) for k, v in sorted(feature\_dict.items(), key=lambda item: item[1], reverse=True)]

[('petal\_width', 0.8954110185300336),

('petal\_length', 0.10458898146996648),

('sepal\_length', 0.0),

('sepal\_width', 0.0)]

petal\_width 변수가 정답을 예측하는데 있어 가장 중요한 역할을, 그리고 petal\_length가 두 번째로 중요한 역할을 하는 것을 알 수 있습니다.

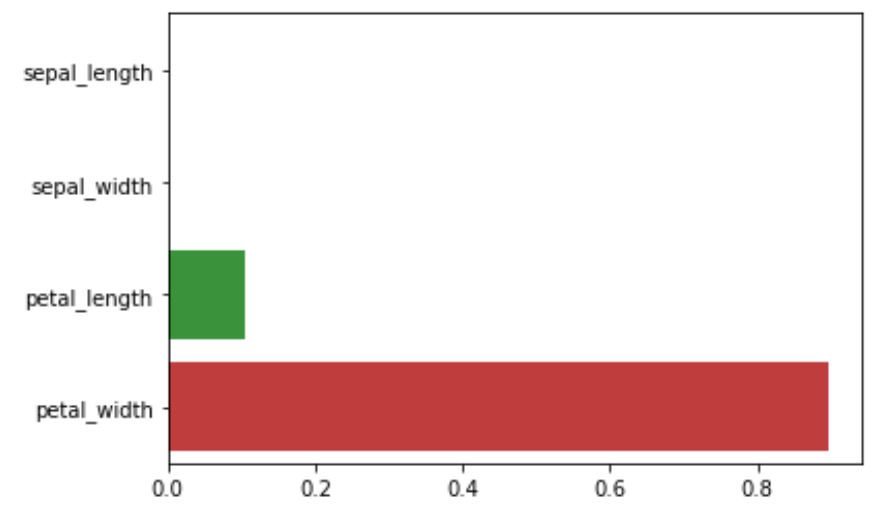
그림 XX를 보면 petal\_width가 제일 먼저 분할 기준으로 사용되었고, 그 다음에 petal\_length가 사용된 것을 확인할 수 있습니다. 그리고 sepal\_length와 sepal\_width는 분할 기준으로 사용되지 않았습니다.

위 결과를 seaborn 모듈의 boxplot() 함수를 이용해서 아래와 같이 시각화할 수 있습니다.

import seaborn as sns

sns.barplot(x=model.feature\_importances\_, y=X.columns)

결과는 아래와 같습니다.



이번에는 GridSearchCV 클래스를 이용해서 하이퍼파라미터 튜닝을 해보도록 하겠습니다. 앞에서 언급한 것 처럼 결정 트리의 주요한 하이퍼파라미터에는 max\_depth, min\_samples\_split, min\_samples\_leaf, max\_leaf\_nodes가 있습니다. 그리고 에러의 정도를 어떠한 지표를 이용해서 측정할 것인지에 대한 criterion이 있습니다. 이러한 하이퍼파라미터는 DecisionTreeClassifier() 생성자 함수의 파라미터로 지정되어 있습니다. 각 하이퍼파라미터의 값으로 시도하고자 하는 값들을 다음과 같이 설정합니다.

from sklearn.model\_selection import GridSearchCV

params={'criterion':['entropy', 'gini'],

'min\_samples\_split':[2, 3, 4, 5, 6, 8, 10],

'min\_samples\_leaf':[0.01, 0.02, 0.03, 0.04, 0.05],

'max\_leaf\_nodes':[10, 15, 20, 25, 30, 35, 40, 45, 50],

'max\_depth':[2,4,6,8],

}

gs\_model = GridSearchCV(model, param\_grid=params, cv=5)

## Tip 시작 ##

효율적인 하이퍼파라미터 튜닝

한 번에 여러 개의 하이퍼파라미터에 대해 여러 값을 이용하여 튜닝을 하는 경우 시간이 오래 걸리게 됩니다. 튜닝 시간을 줄이기 위해서는 한 번에 소수의 하이퍼파라미터의 값들을 튜닝하는 것이 더 바람직합니다.

## Tip 끝 ##

아래와 같이 학습을 합니다.

gs\_model.fit(X\_train, y\_train)

학습이 끝나면 모형의 성능을 가장 좋게하는 하이퍼파라미터의 값들이 best\_params\_ 변수에 저장되어 있습니다. 아래와 같이 확인합니다.

gs\_model.best\_params\_

{'criterion': 'entropy',

'max\_depth': 6,

'max\_leaf\_nodes': 20,

'min\_samples\_leaf': 0.01,

'min\_samples\_split': 2}

그리고 이러한 베스트 하이퍼파라미터의 값을 갖는 모형은 best\_estimator\_ 변수에 저장되어 있습니다. 다음과 같이 도출할 수 있습니다.

best\_gs\_model=gs\_model.best\_estimator\_

best\_gs\_model의 predict() 함수를 이용해서 평가 데이터에 대하 종속변수의 값을 예측합니다.

y\_preds\_gs = best\_gs\_model.predict(X\_test)

# 위의 결과는 아래와 같습니다

# y\_preds\_gs = gs\_model.predict(X\_test)

best\_estimator\_에 저장되어있는 모형을 이용해서 예측을 하는 것과 gs\_model을 이용해서 예측하는 것은 동일한 결과를 반환합니다. 즉, gs\_model을 이용해서 예측하면 기본적으로 성능이 가장 좋은 모형을 이용해서 예측하게 됩니다.

classification\_report() 함수를 이용해서 모형의 성능을 확인해 보겠습니다.

print(classification\_report(y\_test, y\_preds\_gs))

precision recall f1-score support

0 1.00 0.80 0.89 10

1 0.83 1.00 0.91 10

accuracy 0.90 20

macro avg 0.92 0.90 0.90 20

weighted avg 0.92 0.90 0.90 20

위의 결과에서 볼 수 있는 것 처럼 하이퍼파라미터 튜닝을 통해 얻은 베스트 모형의 성능은 하이퍼파라미터 튜닝을 하지 않은 모형의 결과와 동일합니다. 일반적으로는 하이퍼파라미터 튜닝을 통해 얻은 모형의 결과가 튜닝을 하지 않은 모형의 결과 보다 더 좋게 됩니다. 하지만, 위의 예의 경우는 데이터의 크기가 크지 않아 그 차이가 없는 것으로 나왔습니다.

#### 결정 트리를 이용한 감성 분석

이번에는 결정 트리를 이용해서 감성 분석을 수행해 보도록 하게습니다. 여기서는 한글 영화평 데이터를 사용합니다. 해당 데이터는 Korean\_movie\_reviews\_2016.txt 파일에 저장되어 있습니다. 해당 텍스트 데이터는 다음과 같습니다. 영화평과 감성 레이블은 탭(\t)를 구분자로 구분되어 있습니다. 영화평은 키위 형태소 분석기를 이용해서 전처리가 수행되어 있습니다. 감성 레이블의 경우 0은 부정을 1은 긍정을 의미합니다.

부산 행 때문 너무 기대하고 봤 0

한국 좀비 영화 어색하지 않게 만들어졌 놀랍 1

조금 전 보고 왔 지루하다 언제 끝나 이 생각 드 0

…

다음 코드를 이용해서 해당 파일에 저장되어 있는 데이터를 읽어 옵니다. 첫 번째 열과 두 번째 열의 정보를 불어와서 저장을 한 후에 zip() 함수를 이용해서 둘을 구분해서 별도의 변수들 (즉, texts와 labels)에 저장하고 있습니다. 영화평은 texts 변수에 저장되고, 종속변수의 값은 labels 변수에 저장됩니다.

with open('Korean\_movie\_reviews\_2016.txt', encoding='utf-8') as f:

docs = [doc.strip().split('\t') for doc in f]

docs = [(doc[0], int(doc[1])) for doc in docs if len(doc) == 2]

texts, labels = zip(\*docs) # 둘을 분리해서 별도의 list 변수로 저장

texts 변수의 첫 번째 원소에 해당하는 영화평은 아래와 같습니다.

texts[0]

'부산 행 때문 너무 기대하고 봤'

그리고 해당 영화평의 종속변수 값(0, 부정)은 아래와 같이 확인할 수 있습니다.

labels[0]

0

전체 관측치의 수는 165,384이고 그 중 긍정의 레이블을 같는 영화평의 수는 86,806로 전체의 52.5% 정도가 됩니다. 어느 정도 균형이 맞는 데이터라고 생각할 수 있습니다.

일단 먼저 아래와 같이 학습 데이터와 평가 데이터로 구분합니다. 전체 관측치의 수가 많기 때문에 여기서는 전체 데이터의 10%만 평가 데이터로 사용합니다.

from sklearn.model\_selection import train\_test\_split

train\_texts, test\_texts, train\_labels, test\_labels = train\_test\_split(texts, labels, test\_size=0.1, random\_state=0)

그 다음에는 각 영화평을 벡터로 표현하는 것이 필요합니다. 여기서는 CountVectorizer 클래스를 이용해서 TF 기반의 벡터로, 그리고 TfidfVectorizer 클래스를 이용해서 TF-IDF 기반의 벡터로 변환해 보고 각 벡터 정보를 사용했을 때 모형의 성능이 어떻게 차이가 나는지 비교를 해보도록 하겠습니다. 먼저 CountVectorizer을 이용합니다. 코드는 아래와 같습니다. 아래의 코드를 수행하면 학습 데이터에 존재하는 영화평에 대한 DTM이 tf\_train\_features에 저장되고, 평가 데이터의 영화평에 대한 DTM은 tf\_test\_features에 저장됩니다.

from sklearn.feature\_extraction.text import CountVectorizer

tf\_vectorizer = CountVectorizer()

tf\_train\_features = tf\_vectorizer.fit\_transform(train\_texts) # 학습 데이터의 영화평 벡터 변환

tf\_test\_features = tf\_vectorizer.transform(test\_texts) # 평가 데이터의 영화평 벡터 변환

여기서는 간단하게 아래와 같이 클래스 생성자 함수가 갖는 파라미터들의 기본값을 이용해서 객체를 생성해보도록 하겠습니다.

from sklearn.tree import DecisionTreeClassifier

dt\_clf = DecisionTreeClassifier()

그런 다음 fit() 함수를 이용해서 학습을 하고 predict() 함수를 이용해서 평가 데이터에 존재하는 영화평의 종소 변수 값을 예측합니다.

dt\_clf.fit(tf\_train\_features, train\_labels)

pred\_labels = dt\_clf.predict(tf\_test\_features)

classification\_report() 함수를 이용해서 모형의 성능을 살펴보면 아래와 같습니다.

print(classification\_report(test\_labels, pred\_labels))

precision recall f1-score support

0 0.80 0.80 0.80 7766

1 0.82 0.82 0.82 8773

accuracy 0.81 16539

macro avg 0.81 0.81 0.81 16539

weighted avg 0.81 0.81 0.81 16539

모형의 성능이 나이브 베이즈나 로지스틱 회귀 모형에 비해서 좋지 않은 것을 확인할 수 있습니다. 나이브 베이즈의 경우는 f1 값이 .. 이었고, 로지스틱 회귀 모형의 경우는 f1 값이 … 이었습니다.

이번에는 TF-IDF 정보를 사용해서 모형을 학습하고, 평가 데이터에 대해서 성능을 확인해 보겠습니다. 코드는 아래와 같습니다.

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf\_vectorizer = TfidfVectorizer()

tfidf\_train\_features = tfidf\_vectorizer.fit\_transform(train\_texts)

tfidf\_test\_features = tfidf\_vectorizer.transform(test\_texts)

dt\_clf\_tfidf = DecisionTreeClassifier()

dt\_clf\_tfidf.fit(tfidf\_train\_features, train\_labels)

pred\_labels\_tfidf = dt\_clf\_tfidf.predict(tfidf\_test\_features)

print(classification\_report(test\_labels, pred\_labels\_tfidf))

precision recall f1-score support

0 0.80 0.80 0.80 7766

1 0.82 0.83 0.83 8773

accuracy 0.81 16539

macro avg 0.81 0.81 0.81 16539

weighted avg 0.81 0.81 0.81 16539

TF 정보를 이용하는 경우에 비해서 모형의 성능이 약간 더 좋은 것을 확인할 수 있습니다.

## 앙상블 (Ensemble)

결정 트리의 경우, 일반적으로 하나의 트리만을 사용하게 되면 모형의 성능이 별로 좋지 않습니다. 이의 주요한 이유는, 과적합 문제가 크게 발생하기 때문입니다. 결정 트리의 경우 예측력을 높이기 위해서 보통은 하나의 트리만을 사용하지 않습니다. 많은 경우 여러 개의 트리를 사용하게 됩니다. 그리고 보통 하나의 tree를 하나의 learner라고 표현합니다. 그리고 하나의 learner는 예측력이 떨어지기 때문에 weak learner 라고 합니다. 즉, 여러개의 weak leaner를 합쳐서 하나의 strong한 learner를 만드는 것이 ensemble 방법의 기본 목적입니다.

앙상블 방법은 크게 bagging과 boosting 방법으로 구분됩니다. 각 방법에 대해 살펴보겠습니다.

### Bagging (Bootstrap aggregating)

Bagging은 Bootstrap aggregating을 의미합니다. Bagging 방법은 우리가 갖고 있는 하나의 dataset 혹은 학습 데이터를 이용하여 여러개의 subsample data 를 만들고, 각 subsample을 이용해서 예측값을 계산하고 그 값을 평균으로 최종 예측을 하는 방법입니다 (분류의 문제에서는 mode 값을 사용합니다). 예를 들어서, 학습 데이터에 100개의 data points가 있다라고 가정합시다. 이 100개의 data points를 이용해서, 같은 sample size의 서로 다른 여러 개의 sample을 만들 수가 있을 것 입니다. 그리고, 각 sample에 대해서 DT를 적용해서 종속 변수의 값을 예측할 수 있는 것입니다. 예를 들어서 서로 다른 sample이 10개가 있다고 가정을 합시다. 특정한 독립변수의 값들에 대해서, 그러면 이 됩니다.

Bootstrapping

우리가 실제로 가지고 있는 sample을 이용해서 또 다른 sample을 뽑는 방법으로 bagging에서는 bootstrapping 방법을 사용합니다. bootstrapping 은 resampling 방법의 일종으로 replacement를 허용하면서 resampling을 하는 것입니다.

예) sample size 즉, 가지고 있는 data points의 갯수가 100라고 한다면, 같은 sample size의 dataset을 여러개를 만드는 것입니다.

Original dataset (original sample) =>

sample1 =>

sample2 =>

resampling된 각각의 sample은 sample size는 갖지만, sample에 포함되어 있는 data points는 약간씩 다릅니다. 이는 replacement가 포함되어 있기 때문에 그렇습니다.

=> Bagging을 하게 되면, 예측력이 증가할 뿐 아니라 overfitting 문제가 줄어든다 (low variance)

regression은 평균을, classification은 majority voting 방법을 선택 (가장 많이 나온 클래스를 선택)

각 sample에 DT 알고리즘을 적용하여 학습

새로운 관측치에 대해서 각 샘플에 적용하여 예측치를 도출한다.

In Python,

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html>

Finally, if set to True, oob\_score calculates the out-of-bag score for the base learners.

### Random Forests

Bagging과 유사합니다. 즉, 여러개의 sample을 bootstrapping 방법을 사용해서 구축하고, 각 sample에서의 예측치의 값을 이용하여, 평균을 낸다던지 아니면 majority voting 방법을 사용해서 전체의 예측을 하게 됩니다.

Bagging 방법의 주요 단점

Bagging의 경우, decision node에서 고려되는 독립변수가 subsample 마다 크게 달라지지 않습니다. 즉, 종속변수를 잘 설명하는 독립변수들이 일반적으로 먼저 사용되게 되는 것입니다. 어떠한 독립변수가 종속변수를 잘 설명하느냐는 데이터 마다 달라질 수 있기 때문에, bagging의 경우 각 decision node에서 선택되어지는 독립변수가 학습데이터의 영향을 많이 받는다는 것을 의미합니다. 즉, 과적합 문제가 발생할 가능성이 있습니다.

이러한 문제를 해소하기 위한 것이 Random forest입니다. 각 decision node에서 모든 독립변수를 모두 고려하는 것이 아니라, 일부의 독립변수만을 고려하게 됩니다. 그리고 이러한 독립변수들은 random 하게 선택됩니다.

하지만, 기본적인 bagging 방법과 다른 것은 random forest는 각 decision node에서 데이터를 split하는데 모든 feature를 사용하는 것이 아니라 random하게 선택된 일부의 feature들만을 사용합니다. 보통, for classification => m=, 예, if p = 100, then m = 10. for regression => m = p/3

In random forests, for each tree and at each node, only a subset of the available features is considered when choosing the best feature/split point combination.

순서

1. Select the number of features m that will be considered at each node

2. For each base learner, do the following:   
- Create a bootstrap train sample   
- Select the node to split   
- Select m features randomly   
- Pick the best feature and split point from m   
- Split the node into two nodes

Repeat from step 2-2 until a stopping criterion is met, such as maximum tree depth

**How oob\_score works**

<https://towardsdatascience.com/what-is-out-of-bag-oob-score-in-random-forest-a7fa23d710>

Kyriakides, George; Margaritis, Konstantinos G.. Hands-On Ensemble Learning with Python: Build highly optimized ensemble machine learning models using scikit-learn and Keras (p. 182). Packt Publishing. Kindle Edition.

Extra trees (see the kindle book)

Kyriakides, George; Margaritis, Konstantinos G.. Hands-On Ensemble Learning with Python: Build highly optimized ensemble machine learning models using scikit-learn and Keras (p. 232). Packt Publishing. Kindle Edition. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

**max\_features** : int, float, string or None, optional (default=”auto”)

The number of features to consider when looking for the best split:

* If int, then consider max\_features features at each split.
* If float, then max\_features is a fraction and int(max\_features \* n\_features) features are considered at each split.
* If “auto”, then max\_features=sqrt(n\_features).
* If “sqrt”, then max\_features=sqrt(n\_features) (same as “auto”).
* If “log2”, then max\_features=log2(n\_features).
* If None, then max\_features=n\_features.

Bagging 같은 경우에는 각 decision node에서 모든 feature를 고려해서 data를 split하게 되는데, 이렇게 되면 여러개의 sample, 즉, 여러개의 tree를 사용할 지라도 매번 비슷한 feature들이 (먼저) 사용될 가능성이 높습니다 (즉, 중요한 역할을 하는 feature들만 주로 사용되는 것이지요). 이렇게 되면, 서로 다른 tree들 간의 차이가 별로 없습니다. 즉, 결과에 있어서의 상관관계가 굉장히 큰 것이지요. 이러한 경우에는 여전히 overfitting 문제가 존재하는 것이고, 예측의 정확성도 떨어지게 됩니다. 이러한 문제를 random forest를 사용해서 해결 할 수 있는 것입니다.

See Lior Rokeach (Ensemble Learning)

split을 할 때 마다 subsample을 사용하는 것도 가능한다.

#### Random Forest를 이용한 감성 분석

코드는 RF\_sentiment\_example.ipynb 파일을 참고하세요.

### Boosting

부스팅은 기본적으로 같은 데이터에 대해서 learner 혹은 tree를 순차적으로 (sequentially) 적용하는 것을 의미합니다. 그리고 각 tree 혹은 learner의 목적은 이전 learner가 설명하지 못한 부분을 추가적으로 설명하는 것입니다. Boosting의 방법에는 크게 두 가지 종류가 있습니다. 하나는 gradient boosting이고 다른 하나는 adaptive boosting입니다.

#### AdaBoosting (Adaptive boosting)

DT를 base learner로 사용하는 경우, updated weights가 어떻게 사용되는가?

<https://stats.stackexchange.com/questions/157220/how-to-use-decision-stump-as-weak-learner-in-adaboost> 이게 맞나? 이는 class에 weight를 주는 것 아님?

예

n개의 data points가 있다고 가정하겠습니다. 몇개의 독립변수가 있고 하나의 종속변수가 있고, 해당 종속변수가 취할 수 있는 값이 1과 -1 이라고 가정합시다.

Adaboost는 각 data point마다 가중치를 줍니다. 처음에는 모두 같은 가중치를 갖습니다. data points의 수가 n인 경우에 각 data point가 갖는 weight는 이 됩니다.

어떠한 알고리즘을 사용할지를 결정합니다. DT를 사용할 수도 있고, Logistic regression을 사용할 수도 있습니다. Adaboost에서 사용되는 비용함수는 다음과 같이 표현됩니다.

m 단계에서의 DT는 의 값을 최소화 것으로 결정됩니다. 각 단계에서 를 이용해서 각 데이터 포인트의 종속변수 값을 예측하게 됩니다.

는 m 단계에서의 I 번째 관측치에 대한 예측치 입니다.

이를 이용해서 다음과 같이 예측치를 update 합니다.

은 각 learner에 대한 가중치라고 생각할 수 있습니다. 은 다음과 같이 계산됩니다.

이것 대신

를 사용하는 경우도 있어, = learning rate

이는 각 learner에 대한 비용함수의 값 ()이 작을수록 커짐을 알 수 있습니다. 즉, 예측을 잘한 learner의 가중치를 크게 주겠다라는 뜻입니다.

그리고 m 단계에서의 결과를 가지고 각 data point에 대한 weight를 업데이트 합니다.

즉, 틀린 것은 가중치를 많이주고, 맞은 것은 가중치를 작게 주겠다라는 뜻입니다.

weight가 커진다는 것은 비용함수에서의 비중이 그 만큼 커진다는 것을 의미합니다. 즉, 틀린 경우 비용함수의 값이 상대적으로 더 많이 증가하는 것이지요. 그래서, 비용함수의 값을 최소화하기위해서는 우선적으로 weight의 값이 큰 data point의 값들 부터 그 답을 정확하게 예측해야 합니다.

파이썬 코딩

코드는 AB\_sentiment\_example.ipynb 파일을 참고하세요.

#### Gradient Boosting

파이선 코딩

코드는 GB\_sentiment\_example.ipynb 파일을 참고하세요.

#### XGBoost

Tianqi Chen in 2016 <https://arxiv.org/pdf/1603.02754.pdf>

성능과 속도 개선

성능 개선에 있어서 가장 중요한 것은 규제화 방법을 사용했다는 것

XGBoost에서는 기본 모형으로 결정 트리를 사용합니다. 그리고 Gradient Boosting의 한 종류이기 때문에 기본적으로 작동하는 방식은 Gradient boosting과 비슷합니다. 즉, 이전 모형이 설명하지 못한 부분 (나머지)를 현재 단계에서 적용되는 모형이 설명하는 식으로 종속변수의 값을 예측합니다. 각 노드는 score를 갖습니다. leaf node가 아니라 할 지라도.

일반적으로 앙상블 방법의 종속변수 예측치는 다음과 같이 표현될 수 있습니다.

여기서 K는 트리의 수, 는 함수 공간 (여러 개의 함수들의 집합 이라고 생각할 수 있습니다). 그리고 는 번째 트리에 대한 함수가 됩니다. 함수라는 것은 트리 자체 (트리의 구조, 노드의 수 등)가 되며, 트리를 통해 예측되는 종속변수의 예측치라고 생각할 수도 있습니다. 는 번째 관측치의 피쳐 정보 벡터입니다.

그리고 그래디언트 부스팅의 경우, t 단계까지의 종속변수의 예측치는 다음과 같이 표현될 수 있습니다.[[50]](#footnote-52)

즉, t-1 까지의 예측치()에 t 번째에서 사용된 트리가 예측하는 값, 즉 을 더하는 것입니다. 그래디언트 부스팅에서는 t 번째에서 사용된 트리가 예측하는 값, 즉 는 이전 단계 트리가 설명하지 못한 나머지에 대한 예측치가 됩니다. 즉, 에 대한 예측치가 되는 것입니다.

규제화 항을 포함하고 있는 비용함수는 아래와 같이 표현될 수 있습니다.[[51]](#footnote-53) 이는 t 단계에서의 비용함수가 됩니다.

는 모든 관측치들에 대한 원래의 비용함수가 되는 것이고, 는 t 단계까지 적용된 트리들에 대한 규제화 항이 됩니다. 는 모형의 복잡한 정도를 나타낸다고 생각하면 됩니다.

이는 다시 아래와 같이 표현될 수 있습니다.

위 비용함수는 특정 단계에서 사용되는 트리 전체의 비용함수가 되기 때문에, t 단계에서 선택되어져야 하는 트리는 위 비용함수를 최소화하는 트리가 되어야 합니다.

만약 SE (squared errors) 비용함수를 사용한다고 하는 경우, 위 식은 다음과 같이 표현될 수 있습니다.

이를 풀면 아래와 같이 표현될 수 있습니다. 아래에서 는 t에 영향을 받지 않는 값들을 의미합니다. 만 남은 이유도 t-1 단계까지의 트리의 구조는 이미 결정되어 있기 때문에 t에 더 이상 영향을 받지 않기 때문입니다 (다르게 표현하면 t 단계에서 그 구조가 변경되지 않기 때문입니다).

SE의 경우에는 위와 같이 간단하게 표현되지만, 만약 교차 엔트로피 등의 다른 비용함수가 사용되는 경우, 위와 같이 간단한 형태로 표현되지 않습니다. 그래서 원래 비용함수가 무엇인지와 상관없이 비용함수 부분을 간단하게 표현하기 위해서 XGBoost에서는 근사치를 사용합니다. 이를 위해서 테일러 급수 (Taylor series)를 사용합니다 (정확하게는 2차 테일러 급수를 사용합니다)

참고



그러한 경우 목적함수는 아래와 같이 표현됩니다.

여기에서 와 는 아래와 같이 정의됩니다.

는 비용함수 를 에 대해서 한 번 미분한 값을 의미하고 는 두 번 미분한 값을 의미합니다. 두 번 미분한 값을 헤시안 (Hessian)이라고도 표현합니다.

불필요한 상수항을 삭제한 이후, t 단계에서의 목적 함수는 아래와 같습니다.

이것이 t 단계에서 적용되는 트리의 목적함수가 됩니다. 이 목적함수를 이용해서 t 단계에서 적용되는 트리의 노드가 어떻게 분할되는지는 잠시 후에 설명하겠습니다.

위 목적함수의 장점은 원 목적함수가 무엇이든 상관없이 와 값만을 알면된다는 것입니다.

모형의 복잡도

은 모형의 복잡도를 의미한다고 했습니다. 그렇다면 XGBoost에서는 가 어떻게 정의될까요? 이를 위해, 를 조금 더 명확하게 정의해보도록 합니다.

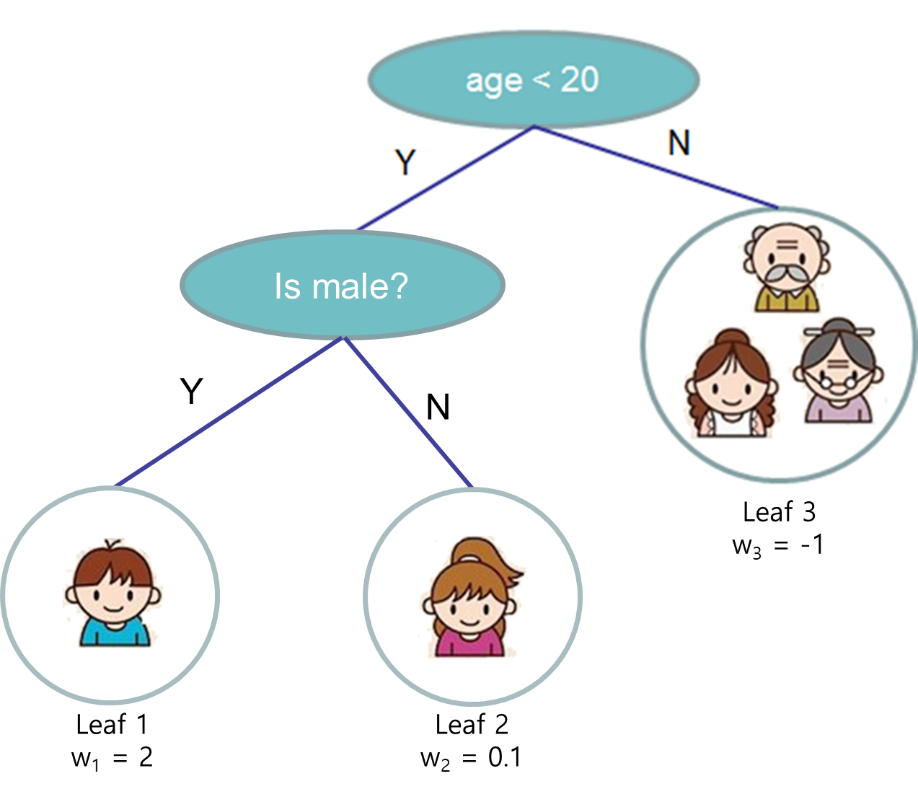
는 특성 정보를 담고 있는 d 차원의 벡터가 됩니다. 그리고 는 t 단계에서 적용되는 트리가 됩니다. 는 리프 노드들의 score의 벡터입니다. 는 각 관측치를 특정 리프 노드에 할당하는 역할의 함수입니다. T는 리프 노드의 수입니다.

복잡도는 아래와 같이 다음과 같이 정의됩니다.

여기서 는 단계에서 적용되는 트리가 갖는 리프 노드들의 score의 벡터가 됩니다. 스코어는 각 노드에서 계산되는 종속변수의 값이라고 생각할 수 있습니다.

여기에서 는 규제화 부분 즉 모형의 복잡도를 나타내는 항이 됩니다. 그리고 는 L2 규제화 항이 됩니다 (각 값의 제곱합으로 표현되기 때문에 그렇습니다). XGBoost에서 제공되는 XGBoostRegressor나 XGBoostClassifier 클래스 생성자 함수가 갖는 reg\_lambda 가 를 의미하고 L2 규제화 파리미터를 의미합니다.

특정한 트리 구조에 대해서 위의 복잡도가 어떻게 계산되는지 구체적인 예를 이용해서 계산해 보겠습니다.



는 각 리프 노드에서 예측되는 종속변수의 값을 의미하고, XGBoost에서는 스코어(score)로 표현됩니다. 즉, Leaf 1에 있는 한 명의 사람 (한 개의 관측치)는 종속변수의 값이 2가 되는 것이고, Leaf 2에 있는 한 명의 사람의 종속변수 값은 0.1, 그리고 Leaf 3에 있는 세 명의 사람들은 -1의 동일한 종속변수 값을 갖게 되는 것입니다. 위에 대해서 모형의 복잡도 즉,

이 됩니다.

저자들에 따르면 모형의 복잡도를 정의하는 방식에는 여러가지가 있을 수 있지만, 위의 정의는 실제 데이터 분석에서 잘 작동한다고 합니다.

는 다시 아래와 같이 표현될 수 있습니다 (로 정의되었다는 것을 기억하세요).

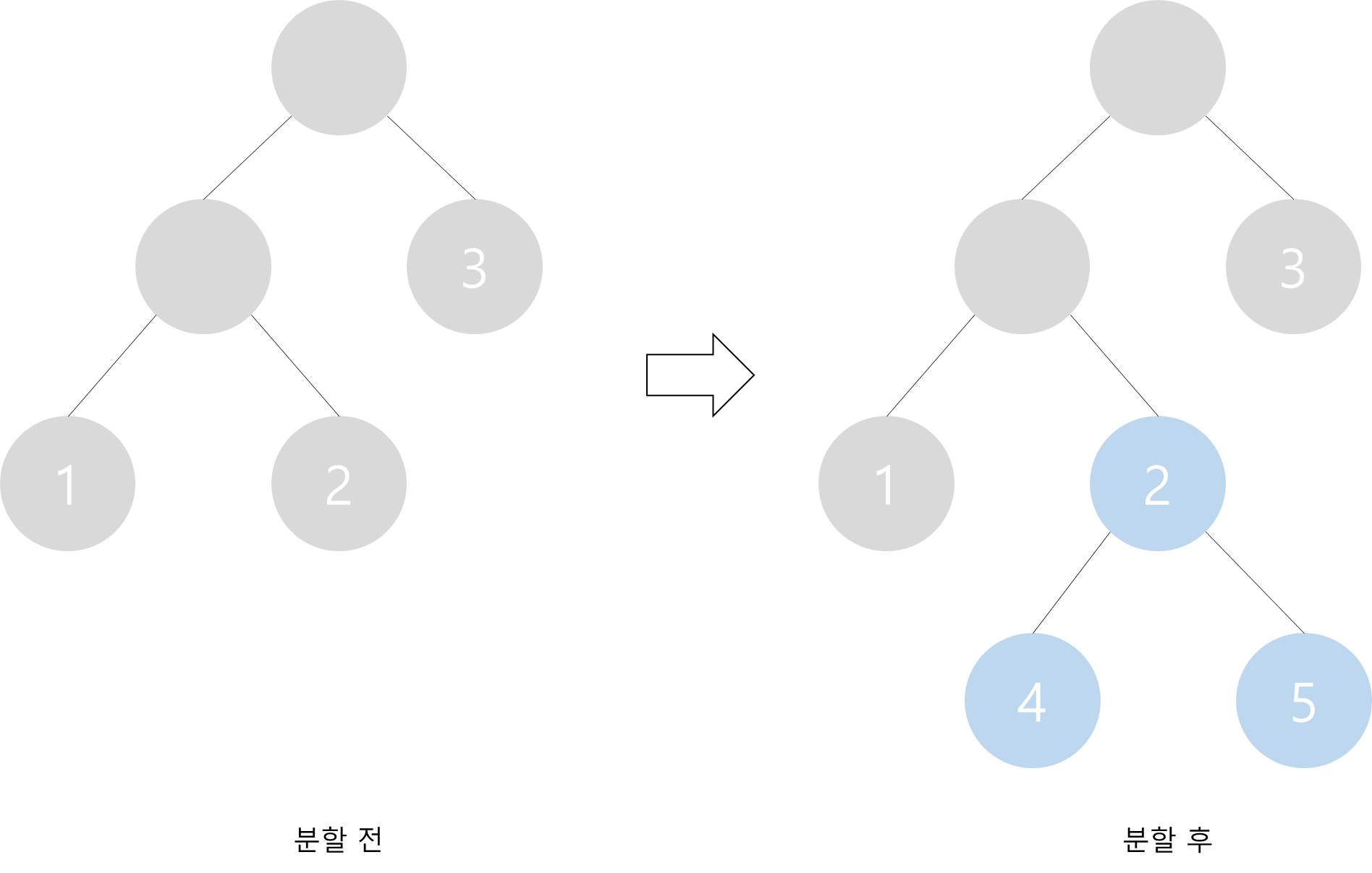
위의 식은 다음과 같이 표현될 수 있습니다. 아래와 같이 표현될 수 있는 이유는 같은 노드에 속한 관측치들은 모두 같은 값(score)을 갖기 때문에 그렇습니다. 그리고 가 됩니다.

다시 한번 강조하지만 T는 트리의 수가 아니라 리프 노드의 수입니다. 는 번째 리프 노드에 속한 관측치 를 의미합니다. 즉, 는 번째 리프 노드에 속한 관측치들의 의 합을 의미합니다. 라고 정의하고, 정의하면 위 식은 다시 아래와 같이 간단하게 표현됩니다.

각 트리의 에 대해서, 이를 에 미분해서 정규 방정식을 풀면 의 최적값이 아래와 같이 도출됩니다.

를 다시 위식에 대입하면 아래와 같은 목적함수가 도출됩니다 (Minimum objective function 이라고도 합니다).

그렇다면 위의 목적함수는 결정 노드에서 서로 다른 두 개의 그룹으로 분할할 때 어떻게 사용될까요? 설명을 위해서 아래의 그림을 참고하겠습니다.



분할 전의 리프 노스의 수는 3 (노드 1, 2, 3)이고, 분할 후 리프 노드의 수는 4 (노드 1, 3, 4, 5)입니다.

분할 전의 목적함수 값은 다음과 같이 됩니다.

노드 2가 노드 4와 5로 분할 되었을 때 발생하는 Gain은 분할 전과 분할 후의 목적 함수 값으 차이로 정의됩니다 (즉, ). 이는 다음과 같습니다.

하나의 노드가 왼쪽 노드와 오른쪽 노드 두 개로 분할이 되는 경우 발생하는 Gain은 아래와 같이 일반화될 수 있습니다.

이 값을 가장 크게하는 특정 정보와 해당 정보의 값을 이용해서 두 개의 노드로 분할이 됩니다. 그리고 Gain의 값이 음수가 되는 경우, 노드 분할을 중단합니다. 즉, 의 값이 크면 클수록 Gain의 값이 음수가 될 가능성이 더 커지게 때문에 분할을 더 적게하게 됩니다. 즉, 모형이 상대적으로 단순해 지고 따라서 과적합 문제가 발생할 가능성이 줄어들게 됩니다 (는 하이퍼파라미터로 XGBoost 모듈의 gamma 파라미터를 의미합니다).

규제화 부분에 L1 항을 포함하면 아래와 같이 됩니다.

XGBoost 모듈이 갖는 즉, 생성자 함수의 파라미터 중에서 gamma는 위의 를, reg\_lambda는 위의 를, reg\_alpha는 를 의미합니다.

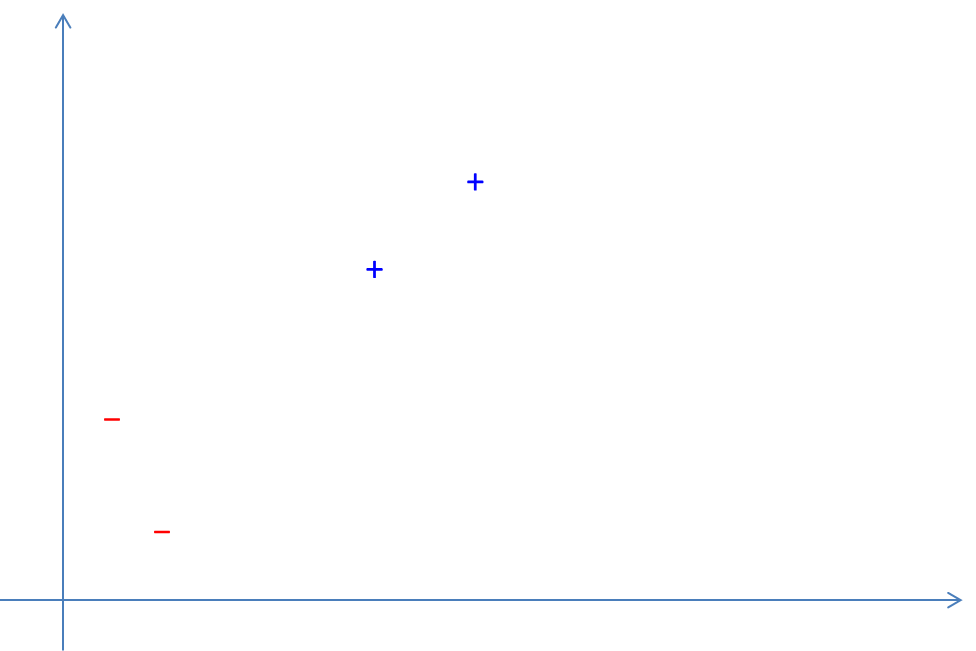
코드는 XGBoost\_sentiment\_example.ipynb 파일을 참고하세요.

#### Light GBM과 CatBoost

## SVM (support vector machines)

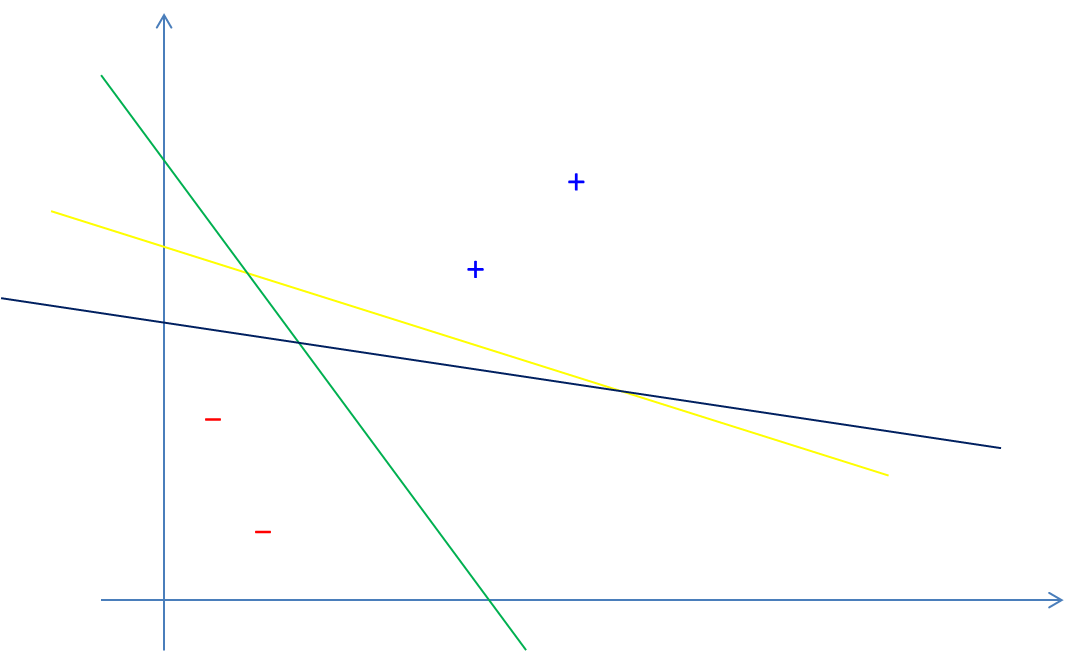
이번에는 SVM을 이용해서 감성분석을 해보도록 하겠습니다. SVM은 수학적으로 꽤 복잡한 알고리즘이기 때문에 본 글에서는 SVM의 기본적인 원리만을 수학적인 내용을 되도록 배제하고 설명합니다. 수학적 내용이 궁금한 독자들은 James et al. (2017), Introduction to Statistical Learning 또는 Statnikov et al. (2011), Gentle introduction to support vector machine in Biomedicine 등의 책을 참고하길 바랍니다.

SVM은 데이터의 분류를 위해 데이터 포인트들을 벡터로 변환하여 특정 차원 공간에 위치 시킵니다 (감성분석에서의 데이터 포인트는 문서가 됩니다). 각 데이터 포인트는 독립변수 혹은 feature 정보를 이용하여 벡터로 변환됩니다. 문서의 경우는 최종적으로 선택되어진 단어 정보들을 사용해서 하나의 문서를 벡터로 변환하게 됩니다. 만약 최종적으로 선택되어진 단어들, 즉, vocabulary에 있는 단어들의 수가 1000개라고 한다면, 각 문서는 1000개의 원소를 갖는 벡터로 변환됩니다. 그리고 이러한 벡터는 1000차원 공간에 존재하는 하나의 점 (point)로 표현되는 것입니다. SVM은 이렇게 특정 차원의 공간에 존재하는 데이터 포인트들을 종속 변수값에 따라 구분하는 역할을 합니다. SVM 설명을 위해서 데이터 포인트들이 아래와 같이 2차원 공간에 위치한다고 가정하겠습니다.

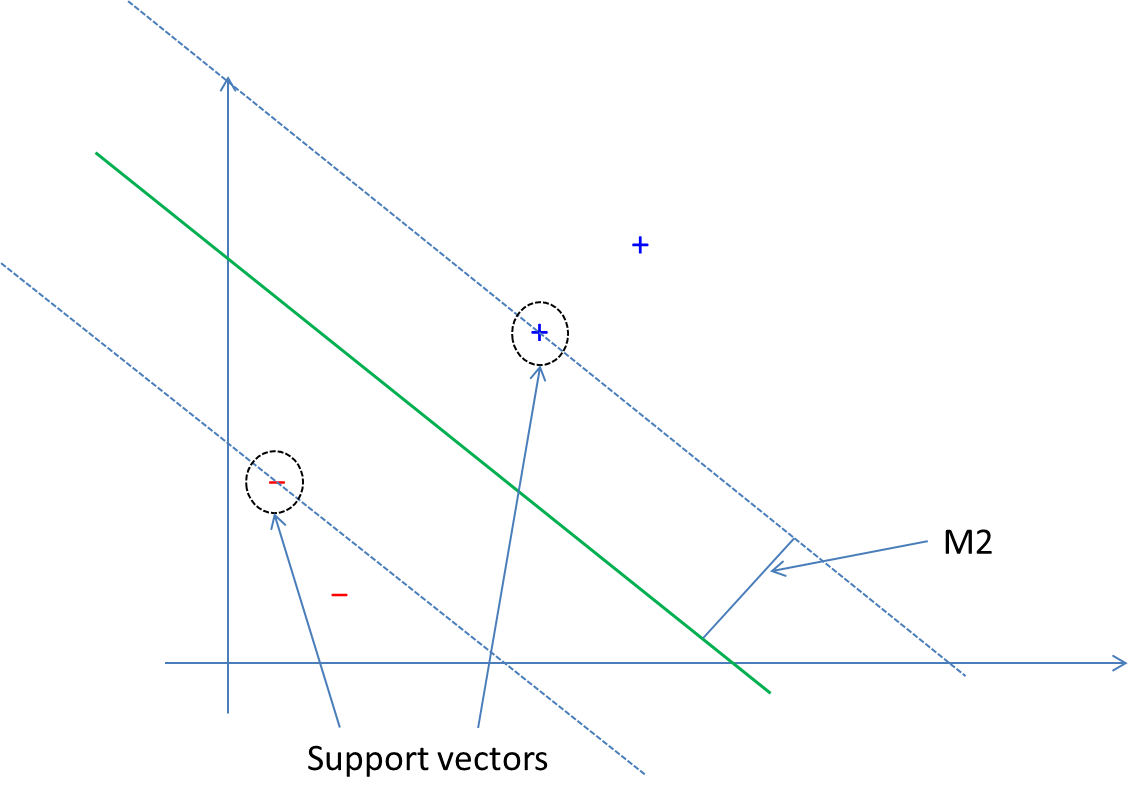
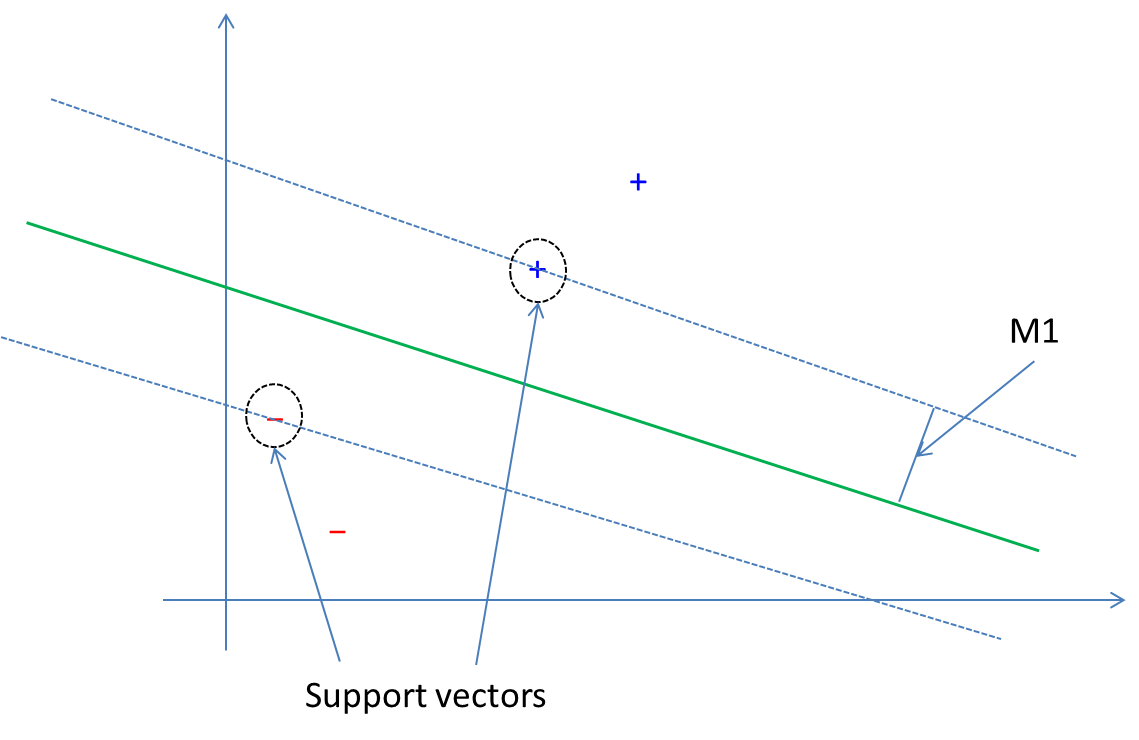


위의 그림에는 총 4개의 관측치 (데이터 포인트)가 있습니다. 그 중에 두개는 +에 해당하는 레이블 (즉, 종속변수 값)을 그리고 나머지 두개는 –에 해당하는 레이블 정보를 가지고 있습니다. 감성분석에서 +는 긍정의 감성이라고 생각할 수 있고, –는 부정의 감성이라고 생각할 수 있습니다.

SVM이 하는 것은 데이터 포인트들을 위와 같이 벡터로 표현하여 특정한 차원의 공간상의 점으로 표현한 다음 종속변수 값 (여기서는 +와 – )에 따라 구분 하는 것입니다. 구분을 할 때 데이터 포인트들을 종속변수의 값에 따라 분류할 수 있는 hyperplane이라고 하는 것을 사용합니다. 2차원 공간상에서의 hyperplane은 하나의 직선이 됩니다 (3차원 공간상에서의 hyperplane은 하나의 평면이 됩니다. 사실 n 차원 공간에서의 n-1차원에 해당하는 hyperplane이 됩니다). 하지만, 위의 그림에 존재하는 4개의 데이터 포인트를 (레이블에 따라서) 나눌 수 있는 직선 (즉, hyperplane)에는 굉장히 많은 것이 존재합니다. 아래와 같이 표현될 수 있습니다.



SVM은 데이터를 분류할 수 있는 이러한 여러개의 hyperplane중에서 최적의 hyperplane을 찾는 작업을 합니다. 그러면 어떠한 hyperplane이 최적이 될까요? 위의 그림과 같은 데이터 포인트들을 구분하는 최적의 직선은 무엇일까요? **데이터 포인트들을 구분하는 직선과 가장 가까이 있는 데이터 포인트들, 즉 벡터들과 거리가 가장 크게 하는 직선**을 찾게 됩니다. 아래 그림을 보면서 설명하겠습니다.



초록색의 직선이 hyperplane이 됩니다. 그리고 hyperplane과 가장 가까이 위치해 있는 벡터들을 support vector라고 합니다. hyperplane과 support vector까지의 거리를 보통 margin이라고 표현을 하는데, 우리는 여러개의 가능한 hyperplane 중에서 이 margin의 크기를 가장 크게 하는 hyperplane을 찾게 됩니다. 만약 위의 그림에서 오른쪽 경우의 margin (M2)이 왼쪽의 margin (M1)보다 더 크다면 왼쪽 보다는 오른쪽에 있는 hyperplane이 더 선호되는 것입니다. 그 이유는 과적합 문제를 줄이고, 새로운 데이터에 대한 예측의 정확성을 높이는 효과가 있기 때문입니다.

2차원 공간에 존재하는 직선의 hyperplane은 다음과 같이 정의 됩니다.

여기에서 과 는 해당 hyperplane 위에 있는 임의의 점이 됩니다. 그리고 해당 hyperplane의 구체적인 위치는 에 의해서 결정됩니다. 즉, 우리는 margin을 가장 크게하는 을 찾아야 하는 것입니다. 해당 값을 로 표현할 수 있습니다. 즉, 학습을 통해서 의 값을 찾게 됩니다. 그리고 새로운 데이트 포인트 에 대해서 해당 데이터 포인트가 hyperplane의 어느 쪽에 위치하는지를 파악하고 그 결과에 따라 의 레이블을 결정하게 됩니다. 즉, 인 경우에는 긍정으로 (위의 예에서는 +로), 인 경우에는 부정으로 (위의 예에서는 –로) 레이블을 결정하게 되는 것입니다.

**값 찾기**

Hyperplane을 결정하는 값은 다음 식을 만족하는 값으로 결정됩니다.

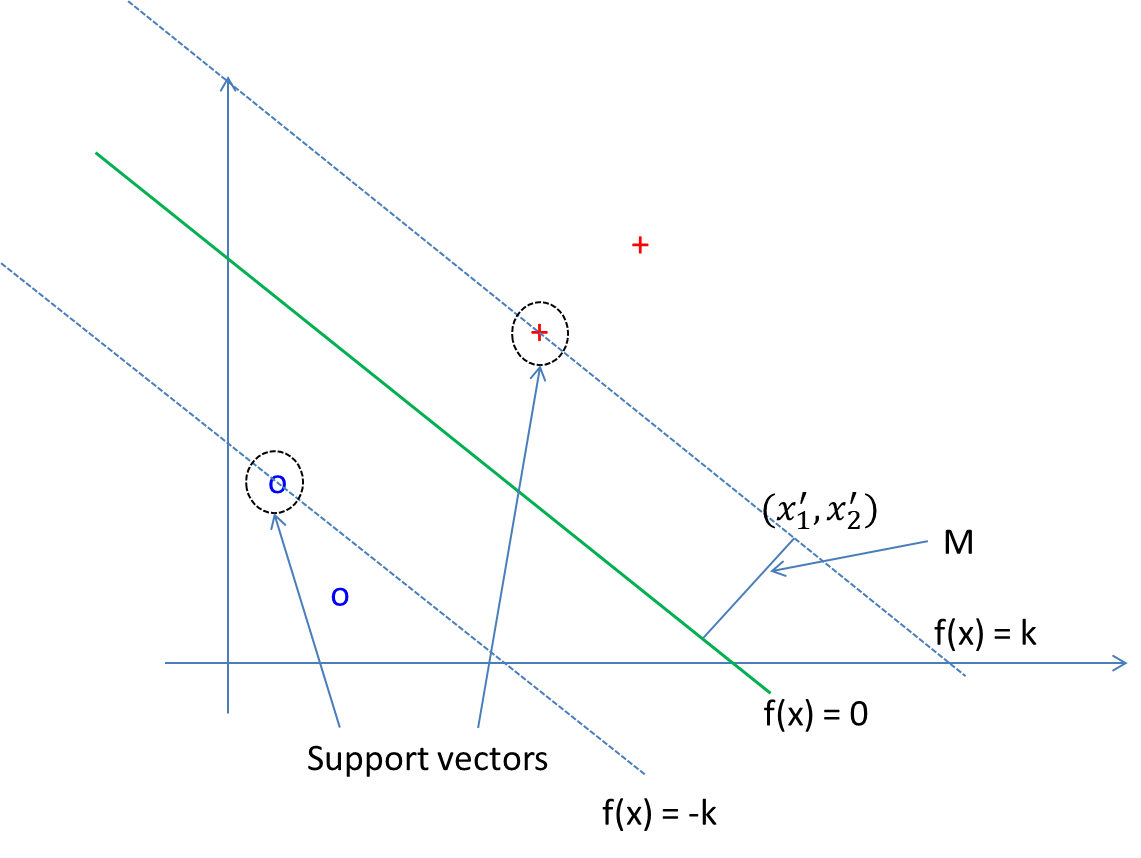
여기에서 M은 margin을 의미합니다. 즉, margin을 가장 크게 하는 값들을 찾는다는 것입니다. 이 margin은 으로 나타내지는 hyperplane과 평행한 또 다른 직선 간의 거리라고 할 수 있습니다. 혹은 hyperplane과 평행한 직선상에 있는 점과 hyperplane 간의 거리라고 할 수 있습니다. 점과 직선 사이의 거리가 어떻게 구해지는지 알아보도록 하겠습니다.

**점과 직선사의 거리**

점()과 직선 ax + by + c = 0 사이의 거리는 다음과 같이 정의됩니다.

라고 표현할 수 있는데, 여기서 점()은 직선 위이 점이 되는 것이고, 이 직선 ()은 으로 표현되는 hyperplane과 평행한 직선이 됩니다. 그리고 점 ()은 해당 직선 상의 한 점이 됩니다.

Hyperplane이 으로 정의되는 경우 아래 그림과 같이 표현될 수 있습니다.



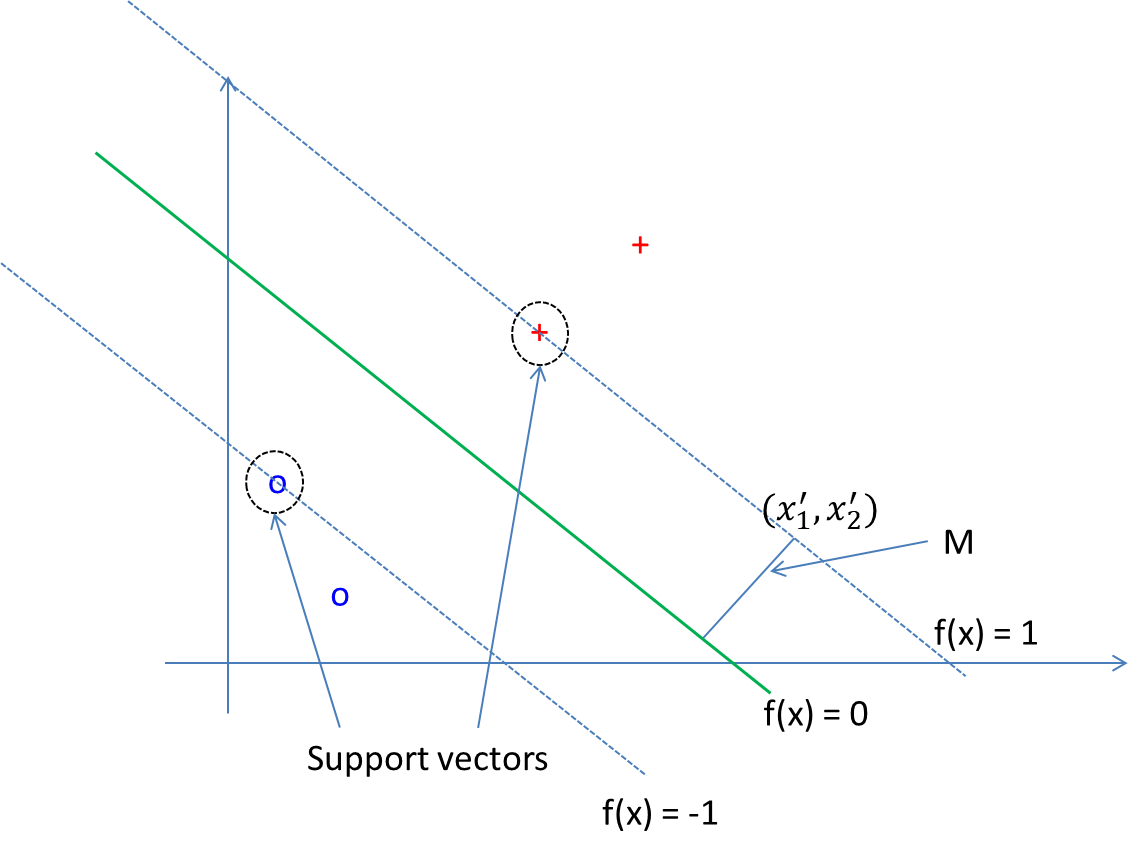
위의 그림에서 가 됩니다. 빨간색의 support vector를 지나는 직선의 방정식이 가 되기 때문에, 해당 직선위에 있는 임의 점 과 hyperplane 직선 간의 거리는 다음과 같이 됩니다.

여기에서 이므로 위의 식은

됩니다. 즉, margin은 다음과 같이 표현됩니다.

이를 최대화하는 과 의 값을 찾아야 하는 것입니다. 그리고 그러한 과 값은 hyperplane을 정의하게 됩니다. k는 상수이기 때문에 위의 식을 최대화 하는 것은 분모인 을 최소화하는 것과 같은 문제입니다. 즉, 다음을 만족하는 b값들을 찾아야 합니다.

최소화문제는 k의 값이 얼마인지와 상관없기 때문에 설명을 용이하게 하기 위해서 여기서는 k=1이라고 하겠습니다. 그러면 support vectors를 지나는 직선은 과 로 표현됩니다 (아래 그림 참고).



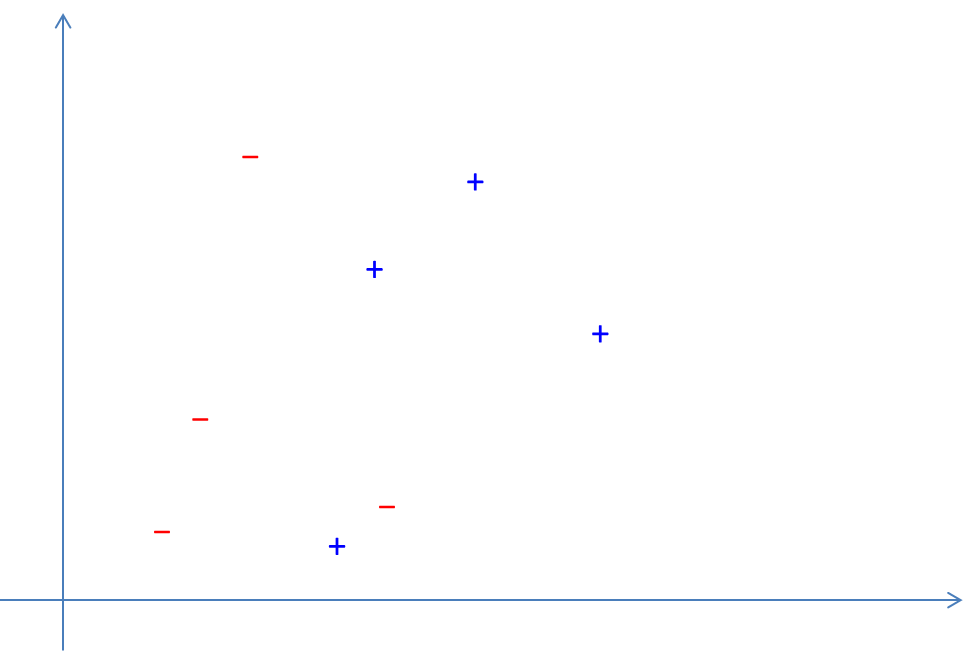
위의 최소화문제를 푸는데 있어 제약 조건이 하나 있습니다. 데이터 포인트가 취할 수 있는 종속변수의 값을 라고 하고 가 취할 수 있는 값에는 -1과 1이 있다고 가정합니다 (즉, ). -1은 부정을 1은 긍정의 레이블을 의미한다고 생각할 수 있습니다. 위의 그림에서 를 만족하는 점, 즉, 데이터 포인트는 긍정의 레이블을 갖고, 을 만족하는 데이터 포인트는 부정의 레이블을 갖는 것입니다. 이 조건은 다음과 같이 표현됩니다.

가 1인 경우 는 1 보다 크거나 같고, 가 -1인 경우 는 -1 보다 작거나 같으므로 와 의 곱은 항상 1 이상이 됩니다. 즉, 우리는 위 조건을 만족 시키는 b값들 중에서 를 최소화 하는 값을 찾아야 하는 것입니다. 이는 아래와 같이 표현될 수 있습니다.

, subject to

**Non-linearly separable cases**

하지만, 항상 데이터 포인트들을 종속변수 값에 따라서 선형 hyperplane을 가지고 구분할 수 있는 것은 아닙니다. 즉, 다음과 같은 상황이 있을 수도 있습니다.



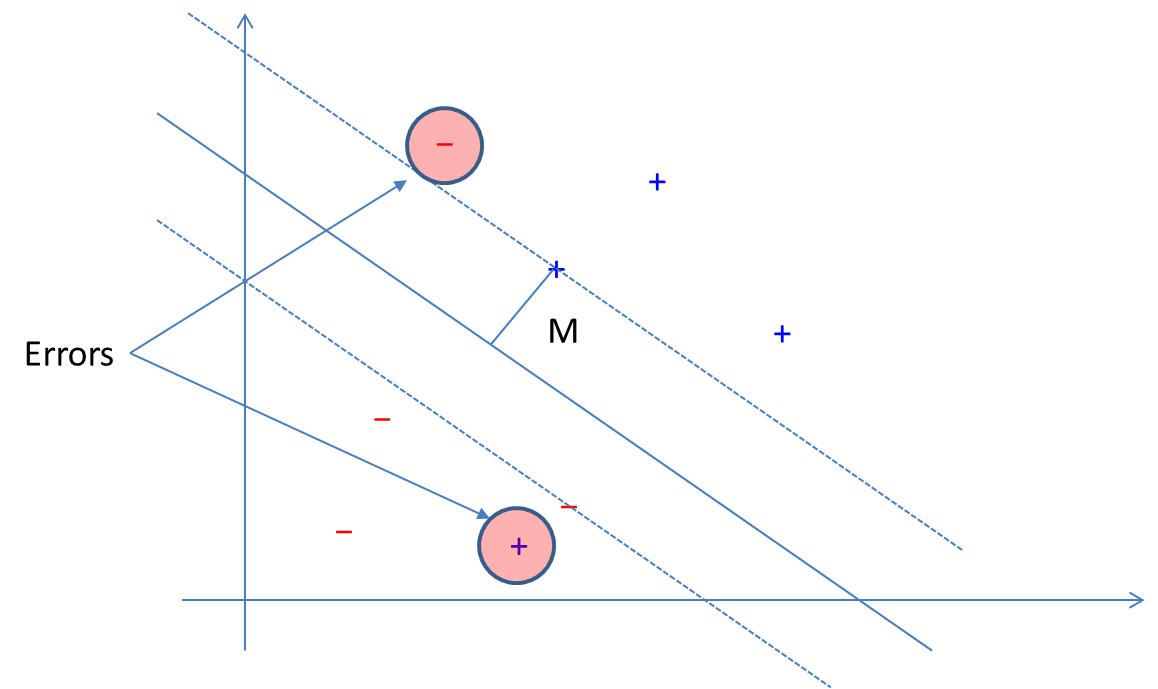
이러한 경우에는 긍정과 부정 데이터 포인트들을 완변하게 구분할 수 있는 선형 hyperplane (즉, 직선)이 존재하지 않습니다. 이러한 경우에는 보통 다음 두가지 방법을 사용하여 문제를 해결합니다.

① Slack 변수 사용하기

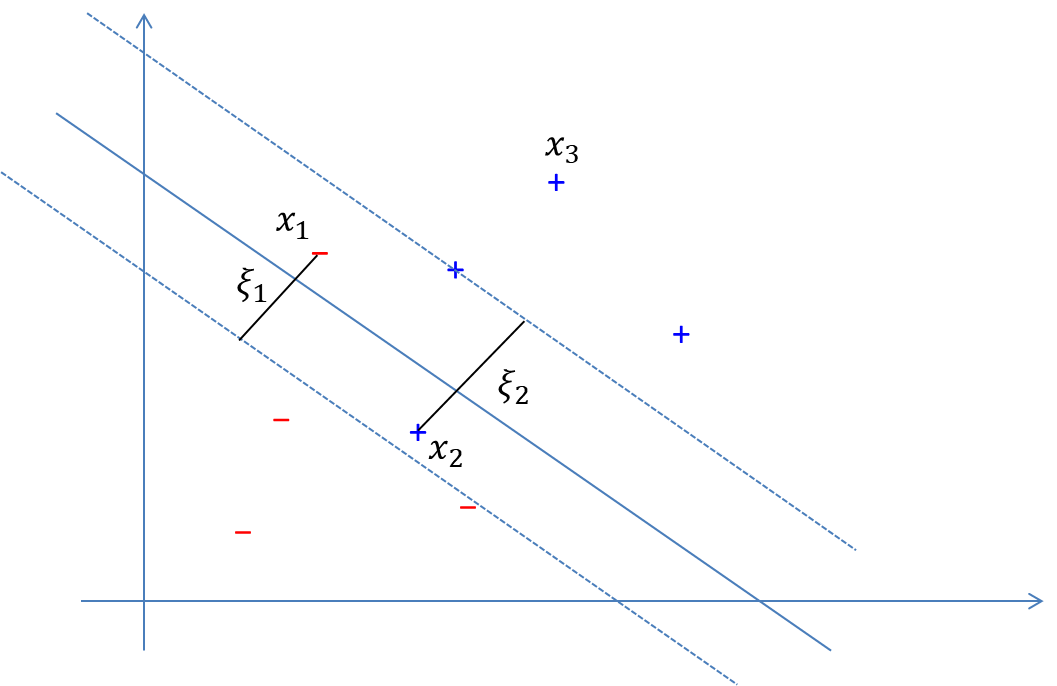
② 데이터 포인트들을 고차원 공간으로 이동시켜 분리하기

① Slack 변수 사용하기

Slack변수를 사용하는 방법은 어느 정도 에러를 인정하면서 분류를 하는 방법입니다. 즉, hyperplane을 찾을 때 잘못 레이블링이 되는 관측치의 발생을 어느 정도 허용하는 방법입니다 (아래 그림 참고).



이를 위해서 slack 변수라고 하는 것을 사용합니다. 조금 더 정확하게 말하면, 각 관측치에 slack 변수 를 할당(assign)합니다. Slack 변수는 각 관측치의 에러 정도를 나타내는 역할을 한다고 생각할 수 있습니다. 분류가 제대로 되는 관측치들의 slack 변수값은 0이 되고, 분류가 제대로 되지 않는 관측치가 갖는 slack 변수의 값만 0보다 크게 됩니다. 분류가 잘못된 관측치의 slack 변수값은 해당 관측치가 (또는 )로부터 떨어진 거리가 됩니다. 위의 예에서 원래 + 클래스를 갖는데 – 클래스로 분류된 관측치의 경우, 해당 관측치의 slack 변수값은 해당 관측치가 로부터 떨러진 거리를 의미하고, 반대로 원래 – 클래스를 갖는데 + 클래스로 분류된 관측치의 경우, 해당 관측치의 slack 변수값은 해당 관측치가 로부터 떨어진 거리를 의미하게 됩니다. 아래 그림과 같이 표현될 수 있습니다. 아래 그림에서 분류가 제대로된 관측치들(예, )의 slack 변수값은 0이 됩니다.



Slack 변수를 사용하게 되면 우리가 풀어야할 최소화 문제가 다음과 같이 됩니다.

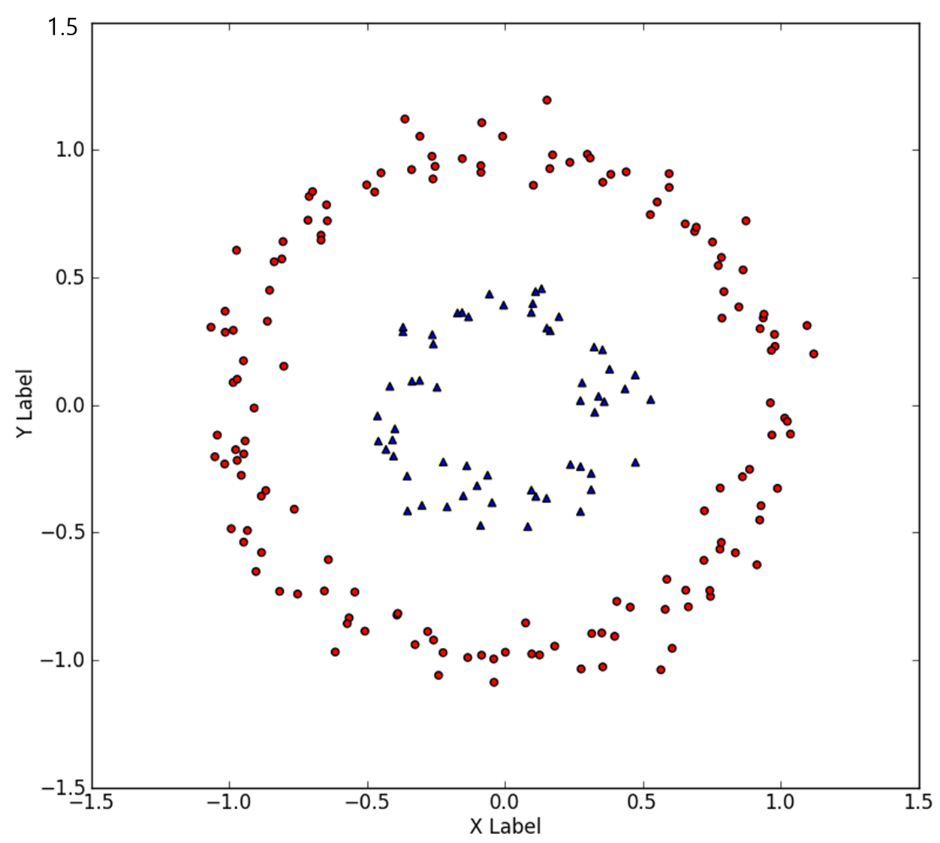
, subject to

위의 목적함수는 margin을 되도록 크게하면서 에러의 정도 (각 관측치들의 slack 변수값의 합으로 표현)를 최소화하는 hyperplane을 찾는다는 것을 의미합니다. 그리고 제약조건 은 잘못 분류되는 관측치들의 경우, (혹은 ) 로부터 만큼 떨어져 있을 수 있다(즉, 에러를 어느정도 허용한다)라는 것을 의미합니다.

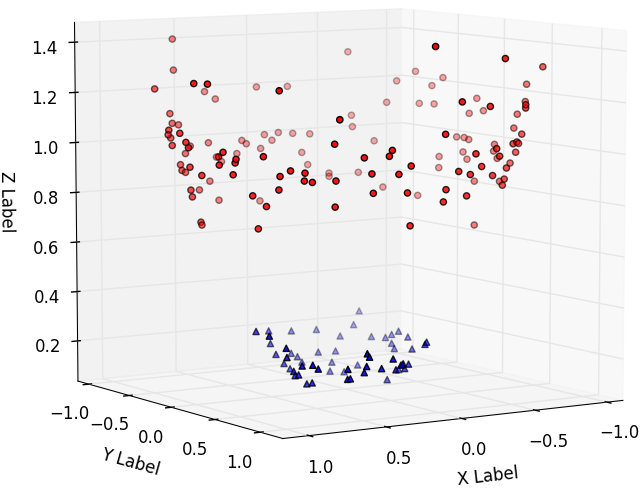
위의 목적함수에서 C는 사용자가 그 값을 정하는 하이퍼파라미터가 됩니다. C의 값이 커지면, slack 변수의 값이 작아져야 한다는 것을 의미합니다. 즉, 학습데이터에 존재하는 에러의 정도가 줄어든다고 생각할 수 있습니다. 하지만, 그렇게 되면 과적합 문제가 발생할 가능성이 높아질 수 있기 때문에 C의 값을 적절하게 선택하는 것이 필요합니다.

**② 데이터 포인트들을 고차원 공간으로 이동시켜 분리하기**

아래와 같이 데이터들이 분포되어 있는 경우에는 선형의 hyperplane을 이용해서 데이터를 구분하는 것이 불가능합니다.



이러한 경우에는 데이터 포인트들을 고차원으로 이동시켜 데이터 포인트들을 구분할 수 있습니다 (아래 그림 참고).



위의 경우에는 빨간색의 데이터 포인트들과 파란색의 데이터 포인트들을 구분할 수 있는 선형의 hyperplane 이 존재합니다 (해당 hyperplane는 평면 형태입니다).

저차원의 데이터 포인트들을 고차원으로 이동시킨후 해당 공간에서 hyperplane을 찾는 과정에는 두 단계가 필요합니다. 1) 데이터 포인트들을 고차원으로 이동시키기, 2) 고차원에서의 데이터 포인트들 사이의 내적 계산하기 (자세한 설명은 본 책의 범위를 벗어나니 Statnikov et al. (2011) 등의 책을 참고하기 바랍니다)

하지만, 데이터 포인트가 많은 경우에 모든 점들을 일일히 고차원으로 이동시킨 다음 데이터 포인트들의 내적을 구하는 데 많은 시간이 소요되기 때문에, SVM에서는 Kernel 방법을 사용합니다. Kernel이라고 하는 것은 선형대수에서 사용되는 개념으로 kernel 함수를 사용하여 임의의 두개의 벡터를 특정 고차원으로 이동 시킨후 그 공간에서의 내적을 구하는 방법입니다. 다만, Kernel 함수를 사용하면 점들을 직접적으로 고차원 공간으로 보내지 않고 간단하게 고차원 공간에서의 내적을 계산한 결과를 얻을 수 있습니다. 자주 사용되는 Kernel 함수의 종류에는 다항 kernel (polynomial kernel), rbf(radius basis function) kernel등이 있습니다.

다항 kernel

서로 다른 두 벡터 (예, )에 대한 다항 kernel의 일반적인 형태는 다음과 같습니다.

는 두 벡터의 내적을 나타냅니다. 그리고 b와 (gamma), d는 하이퍼파라미터가 됩니다. b는 다항 kernel의 계수 (coefficient)라고 합니다. Kernel 함수가 어떻게 작동하는지를 간단하게 설명하기 위해서 여기서는 b = 0, = 1, d=2 이라고 가정하겠습니다. 그러면 다항 kernel은 아래와 같이 됩니다.

인 경우에

이 됩니다. 이는 다르게 표현하면 아래와 같은 두 벡터 간의 내적으로 표현될 수 있습니다.

즉, 는 2차원 공간에 있는 벡터들을 3차원 공간으로 이동 후 해당 공간에서 내적을 계산하는 역할을 합니다. 하지만, 이러한 작업을 하기 위해서 명시적으로 벡터들을 2차원에서 3차원으로 이동하지 않습니다. 그러한 과정없이 3차원에서의 내적값을 구할 수 있는 것입니다. d의 값이 증가할 수록 더 높은 차원에서의 내적값을 구하게 됩니다. 그리고 값이 커지면 고차원 공간에서의 두 벡터의 내적값이 커진다는 것을 알 수가 있습니다. 내적값은 벡터간의 유사도를 반영하므로 내적값이 커진다는 것은 두 벡터가 더 유사한 벡터로 간주된다고 생각할 수 있습니다.

rbf kernel

다항 kernel과 함께 많이 사용되는 것이 rbf kernel이 됩니다. rbf kernel은 아래와 같은 형태를 갖습니다.

여기에서 를 로 대체해서 아래와 같이 표현하는 경우도 있습니다.

여기에서 는 두 벡터간 유클리디안 거리의 제곱을 의미합니다. 의 값이 클수록 고차원에서의 내적값이 작아지는 것을 알 수 있습니다. 즉, kernel 함수의 값은 고차원[[52]](#footnote-54)에서의 두 벡터간의 유사도를 의미하므로 값이 커질수록 벡터 간의 유사도가 작아진다는 것을 의미합니다. 다르게 표현하면 가까이 위치한 벡터들만 유사한 벡터로 간주한다고 볼 수도 있습니다.

**Python code**[[53]](#footnote-55)

전처리 과정과 문서의 벡터화 과정은 로지스틱 회귀 방법에서 설명한 것과 동일합니다. SVM을 이용한 분류기(classifier)는 sklearn에서 제공되는 SVC라는 클래스를 사용합니다. 해당 클래스의 생성자 함수의 경우에는 여러개의 파라미터를 가지고 있는데, 그 중에서 중요한 몇 가지에 대해서 설명하겠습니다.

아래는 sklearn에서 제공하는 해당 생성자 함수의 정의입니다.[[54]](#footnote-56)

class sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache\_size=200, class\_weight=None, verbose=False, max\_iter=-1, decision\_function\_shape='ovr', break\_ties=False, random\_state=None)

C: 첫번째로 중요한 파라미터가 C입니다. C는 앞에서 살펴본 SVM에서 사용하는 목적함수에 포함된 하이퍼파라미터입니다. C의 값이 커질수록 학습 데이터에 대해서 에러를 허용하는 정도가 작아집니다. C가 너무 크면 과적합 문제가 발생할 수 있습니다.

kernel: kernel 파라미터는 SVC에서 사용하는 kernel 함수을 의미합니다. 앞에서 설명한대로 kernel 함수는 저차원의 벡터들에 대해서 고차원으로 이동후의 벡터들 간의 내적을 계산할 때 사용되는 함수입니다. sklearn에서 제공되는 kernel 함수에는 linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ 등이 있습니다. 이중에서 일반적으로 'poly'와 'rbf' 함수가 많이 사용됩니다. 기본값은 'rbf' 로 되어 있습니다.

degree: 사용하는 kernel function이 'poly'인 경우에 지정하는 파라미터입니다. 즉, 각 변수의 최고차항을 결정하는 파라미터가 됩니다. 기본값은 3으로 되어 있습니다. degree의 값이 커질수록 하나의 데이터 포인트가 이동되는 공간의 차원수가 커진다고 생각할 수 있습니다.

gamma: gamma 파라미터는 SVM에서 사용되는 kernel과 관련된 것으로, 사용되는 kernel 함수에 의해 계산되는 고차원에서의 벡터 간의 유사도를 조절하는 역할을 합니다. 예를들어, rbf kernel의 경우 gamma의 값이 작으면 작을수록 고차원에서의 유사도 값이 커지게 됩니다. 즉, 멀리 떨어져 있는 벡터들도 유사도가 크게 계산되는 것입니다. 이렇게 되면, 멀리 떨러져 있는 벡터들 (즉, 관측치들)도 동일한 종속변수의 값을 갖게 되어, 종속변수의 값을 구분하는 decision boundary가 넓어지게 됩니다. 반대로 gamma의 값이 커지면, decision boudary가 좁아지게 됩니다. 이러한 경우에는 과적합 문제가 발생할 가능성이 커지게 됩니다. SVC의 경우, ‘scale’, ‘auto’, float의 값을 취할 수가 있습니다. float은 0보다 큰 실수를 의미합니다. ‘scale’의 경우는 1 / (n\_features \* X.var())을 사용하고, ‘auto’는 1 / n\_features을 사용하게 됩니다. n\_features은 학습데이터에 존재하는 feature의 수를 의미합니다. X.var()은 학습데이터에 존재하는 모든 feature들의 값들에 대한 분산을 의미합니다. 데이터에 존재하는 분산의 값이 작을수록 gamma의 값이 커지는 것을 확인할 수 있습니다.

SVC 클래스는 다음과 같이 간단하게 사용할 수 있습니다 (여기서도 test\_train\_split() 함수를 이용해서 정답이 있는 영화평 데이터를 8:2의 비율로 학습 데이터와 평가 데이터로 구분한 데이터를 사용합니다). 먼저 fit() 함수를 이용해서 학습 데이터에 대해서 학습을 시킵니다. 여기서는 TF-IDF 정보를 갖는 DTM을 사용했습니다.

from sklearn.svm import SVC

clf = SVC(kernel='rbf')

clf.fit(train\_tfidf\_features, train\_labels)

그리고 아래와 같이 평가 데이터의 영화평에 대해서 predict() 함수를 적용해서 긍/부정을 예측하게 됩니다.

pred\_labels = clf.predict(test\_tfidf\_features)

마지막으로 accuracy\_score() 함수를 이용해서 예측의 정확도를 평가합니다.

from sklearn.metrics import accuracy\_score

print('Accuracy: %.2f' % accuracy\_score(test\_labels, pred\_labels))

## 감성어 사전 기반의 감성 분석 (Lexicon-based sentiment analysis)[[55]](#footnote-57)

이번에는 감성어 사전(lexicon)을 이용한 방법에 대해서 간략하게 알아보도록 하겠습니다. 기계학습 기반의 방법은 학습할 수 있는 학습 데이터가 있는 경우에는 용이하게 사용할 수 있지만, 그렇지 않은 경우에는 사용하기 힘들다는 단점이 있습니다. 그리고 도메인 마다 사용되는 용어와 표현이 다르기 때문에 분석 결과가 도메인의 영향을 받는다는 단점도 존재합니다. 이러한 상황에서 보완적으로 사용할 수 있는 것이 감성어 사전 기반 방법입니다. 사전을 이용하는 경우는 일반적으로 아래와 같은 과정을 거쳐서 감성 분석을 진행하게 됩니다.

① 감성어 사전 준비  
일단 사전을 준비해야 합니다. 영어 같은 경우는 잘 만들어진 감성어 사전이 이미 공개되어 있어 그러한 사전을 사용하는 것이 가능하지만[[56]](#footnote-58), 한글 같은 경우에는 공개되어 있는 제대로 된 감성어 사전이 별로 없습니다. 따라서 만약 감성어사전을 이용해서 한글 텍스트에 대한 감성 분석을 해야하는 경우는 사전을 직접 만들어야 한다는 어려움이 있습니다.

감성어 사전은 보통 단어와 해당 단어의 긍∙부정 점수로 구성되어 있습니다 (예, ‘재미없다: -0.5’, ‘밝다: 0.4’, ‘지루하다: -0.4’, ‘흥미롭다: 0.5’ 등). 좀 더 단순하게는 긍정의 단어들과 부정의 단어들로만 분류된 사전을 만들 수도 있습니다 (아래의 예 참고).

pos\_words = ['happy', 'fun', 'great', 'exciting', 'fantastic']

neg\_words = ['boring', 'disappointing', 'bad', 'unpleasant']

② 문서의 감성 점수 계산하기  
감성어 사전이 준비되었으면 다음 단계는 우리가 가지고 있는 텍스트 데이터에서 사전에 있는 단어들을 추출하여 감성어 사전을 이용해서 각 단어의 감성 점수를 부여하고 이를 이용하여 특정 문서의 감성 점수를 계산하면 됩니다. 각 단어의 점수를 이용하여 문서 전체의 감성 점수를 계산하는 방식에는 여러가지가 있을 수 있습니다. 가장 간단하게는 합을 이용하는 방법입니다.

③ 문서의 긍∙부정 판별하기

각 문서에 부여된 감성 점수를 바탕으로 해당 문서의 긍∙부정을 판별하게 됩니다.

**VADER 감성어 사전을 사용해서 감성 분석 해보기**

nltk의 경우에는 VADER (Valence Aware Dictionary and sEntiment Reasoner)[[57]](#footnote-59) 라는 감성어 사전이 제공됩니다. 해당 감성어 사전을 사용하기 위해서는 아래와 같이 SentimentIntensityAnalyzer 클래스를 import를 해줍니다.

from nltk.sentiment.vader import SentimentIntensityAnalyzer

그리고 생성자 함수를 사용해서 해당 클래스의 객체를 아래와 같이 생성합니다.

sid = SentimentIntensityAnalyzer()

이렇게 하면 sid를 이용해서 SentimentIntensityAnalyzer 클래스에서 제공되는 polarity\_scores() 함수를 사용할 수 있습니다. polarity\_scores() 함수를 사용해서 하나의 문장의 감성 점수를 계산합니다.

ss = sid.polarity\_scores('the movie was great')

ss 에는 다음과 같은 값이 저장되어 있습니다.

{'neg': 0.0, 'neu': 0.423, 'pos': 0.577, 'compound': 0.6249}

여기에서 compound의 값이 감성 분석의 총점입니다. 이 값은 -1 (아주 부정) ~ 1 (아주 긍정)의 분포를 갖습니다. 아래의 코드는 또 다른 예를 보여줍니다.

In: sid.polarity\_scores('the movie was awful')

Out: {'neg': 0.5, 'neu': 0.5, 'pos': 0.0, 'compound': -0.4588}

# Topic modeling

이번 섹션에서는 Topic modeling에 대해서 얘기해 보도록 하겠습니다. Topic modeling은 주어진 텍스트 데이터의 주제를 찾을 때 사용하는 방법입니다. 보통은 텍스트 데이터를 구성하는 각 문서의 주제를 찾을 때 사용합니다. Topic modeling에 사용되는 주요 방법으로는 LSI (Latent Semantic Index), pLSA (probabilistic Latent Semantic Analysis), LDA (Latent Dirichlet Allocation), NMF (Non-negative Matrix Factorization)이 있습니다. LSI와 NMF는 행렬기반의 방법이고, pLSA와 LDA는 확률 기반의 방법이라고 생각할 수 있습니다. 여기서는 일반적으로 더 많이 사용되는 LSI와 LDA를 설명하도록 하겠습니다.

## LSI (Latent Semantic Index)[[58]](#footnote-60) (LSA (latent semantic analysis)라고도 표현합니다)

LSI는 차원축소방법을 사용하여 전체 텍스트 데이터를 구성하는 문서들의 주제를 찾는 방법입니다. 사실 LDA가 2003년에 제안되고 나서는 LSI는 topic modeling에 있어서 사용되는 정도가 많이 줄었습니다. 하지만, 차원 축소 방법을 사용해서 문서의 정보를 어떻게 축약하는지에 대한 방법은 topic modeling 의 목적 이외에도 다양한 목적으로 사용될 수 있어서 자연어 처리 혹은 텍스트 마이닝에서 잘 알아둬야 하는 개념입니다. 본 챕터에서는 LSI를 이해하는데 필요한 차원축소 방법인 고유분해 (eigendecomposition)와 그의 사촌격인 SVD (singular value decomposition)에 대해서 먼저 구체적으로 알아보도록 하겠습니다.

#### 고유분해 (Eigendecomposition)

고유분해는 보통 정사각행렬 (square matrix)을 해당 행렬의 고유값(eigenvalues)와 고유벡터(eigenvectors)를 이용해서 분해하는 것을 말합니다. 다음과 같이 표현됩니다. A라고 하는 정사각 행렬이 있는 경우 아래와 같이 분해됩니다.

여기에서 는 A의 고유벡터를 칼럼으로 갖는 행렬이 되고, 는 A의 고유값을 대각성분으로 갖는 대각행렬이 됩니다. 고유값과 고유벡터가 무엇인지 알아보도록 하겠습니다.

#### 고유값과 고유벡터

행렬의 고유값과 고유벡터는 해당 행렬이 갖고 있는 고유한 특성을 나타내는 값과 벡터라고 생각할 수 있습니다. 수학적으로는 정사각행렬 A에 대해서 다음 방정식을 만족시키는 **영(0)이 아닌 벡터** (즉, )가 A 행렬의 고유벡터가 되고 이 고유값이 됩니다.

A는 nxn 행렬, 는 nx1 벡터, 는 스칼라값 (즉, 하나의 숫자)가 됩니다. nxn 행렬의 고유벡터 (고유값)은 n개 존재합니다.

행렬의 기하학적 의미는 (즉, 공간상에서의 의미) 하나의 벡터를 선형이동 (linear transformation)시키는 것입니다. 간단하게 말하면 하나의 벡터를 같은 차원의 공간의 다른 위치로 이동시키거나, 다른 차원의 공간으로 이동시키는 역할을 하는 것입니다.

고유값과 고유벡터를 직접 구해보겠습니다.

은 아래와 같이 변형됩니다.

여기서 는 nxn 단위행렬입니다. 즉, 대각성분만 1이고 나머지 원소의 값은 0인 대각행렬이 됩니다. 이 식이 0이 아닌 v에 대해서 만족하기 위해서는의 역행렬이 존재하지 않아야 합니다. 의 역행렬이 존재하는 경우에는 위의 식을 만족하는 v는 0벡터 밖에 없습니다. 의 역행렬이 존재하지 않는다는 것은 행렬식의 값이 0이라는 것을 의미합니다. 행렬식의 값이 0이 된다는 사실을 사용하여 고유값과 고유벡터를 구할 수 있습니다.

구체적으로 의 고유값과 고유벡터를 구해보겠습니다.

이 되고, 을 만족하는λ = 2, 6입니다. 즉, 두 개의 고유값을 구한 것입니다. 우리는 각 고유값에 대응하는 고유벡터를 구할 수 있습니다. 각 고유값에 대한 고유벡터를 구하기 위해서는 = 0 을 이용합니다.

v = (x, y)라고 하는 경우, 위의 식은 아래와 같이 표현될 수 있습니다.

λ = 2 의 경우에 대해서 위의 식은 아래와 같이 표현됩니다.

3x + y = 0을 만족하는 모든 x, y 가 λ = 2 에 대한 고유벡터가 됩니다. 즉, 고유벡터는 여러개 나올 수 있습니다. 왜냐하면 고유벡터는 방향성만이 중요하기 때문입니다. 보통은 여러개의 고유벡터들 중에서 그 길이가 1인 고유벡터를 선택하게 됩니다.

λ = 6 의 경우에 대해서 위의 식은 아래와 같이 표현됩니다.

x – y = 0만족하는 모든 x, y가 λ = 6 에 대한 고유벡터가 됩니다. 역시나 마찬가지로, 여러개의 고유벡터들 중에서 그 길이가 1인 고유벡터를 선택하게 됩니다.

#### Numpy를 이용해서 고유값과 고유벡터 구하기[[59]](#footnote-61)

파이썬에서는 numpy 모듈을 이용해서 고유값과 고유벡터를 구할 수 있습니다. 여기서는 위의 예에서 다룬 사각행렬인의 고유벡터와 고유값을 numpy를 가지고 구해보도록 하겠습니다.

import numpy as np

A = np.array([[5,1],

[3,3]])

eigVals, eigVecs = np.linalg.eig(A)

위에서 볼 수 있는 것 처럼 Python에서는 numpy에서 제공되는 linalg 모듈이 가지고 있는 eig() 함수를 사용해서 고유값과 고유벡터를 구할 수 있습니다.

eigVals에는 고유값들이 저장되어 있고, eigVecs에는 고유벡터들이 저장되어 있습니다. 고유값을 아래와 같이 확인해 보면

In: eigVals

array([6., 2.])의 값이 나오는 것을 확인할 수 있습니다. 즉, 우리가 직접 구한 두 개의 고유값을 확인할 수 있습니다.

eigVecs는 각 고유벡터를 열로 갖는 행렬이 됩니다. 첫번째 칼럼이 고유값 6에 대한 고유벡터가 되고 두번째 칼럼이 고유값2에 대한 고유벡터가 됩니다. 아래와 같이 각 고유벡터를 추출할 수 있습니다.

v1 = eigVecs[:, 0]

v2 = eigVecs[:, 1]

각 벡터를 확인해 보면 아래와 같습니다.

In: v1

Out: array([0.707107, 0.707107])

v1의 경우 x-y=0을 만족하는 x, y 값들 중에서 벡터의 길이가 1인 것을 만족하는 (즉, ) x, y 값으로 구성된 고유벡터라는 것을 확인할 수 있습니다. 해당 벡터의 길이는 아래와 같이 norm() 함수를 이용해서 구할 수 있습니다.

In: np.linalg.norm(v1)

아래와 같이 v2 고유벡터의 값도 확인할 수 있습니다. v2 역시나 마찬가지로 3x + y = 0을 만족하는 x, y 값들 중에서 벡터의 길이가 1인 조건을 만족하는 x, y 값으로 구성된 것을 확인할 수 있습니다.

In: v2

Out: array([-0.316228, 0.948683])

고유값과 고유벡터에 대해서 간단하게 살펴보았습니다. 이러한 고유값과 고유벡터를 가지고 하나의 정사각행렬을 서로 다른 여러개의 행렬로 분해하는 것을 고유분해라고 합니다. 그리고 이러한 고유분해가 LSI 방법의 핵심이 됩니다.

앞에서 언급한 것 처럼 정사각행렬은 다음과 같이 (고유)분해될 수 있습니다.

앞에서 구한 eigVecs이 V가 되고, 각 고유값을 대각원소로 갖는 행렬이 가 됩니다. 즉, 아래와 같이 표현됩니다.

np.diag()를 사용해서 대각행렬을 생성할 수 있습니다. 즉, 아래와 같이 하면 행렬을 구할 수 있습니다 (여기서는 편의상 행렬을 D 행렬로 표현했습니다).

D = np.diag(eigVals)

그리고 역행렬은 np.linalg.inv()를 이용해서 구합니다. 그리고 행렬 간의 곱하기 연산은 np.dot() 함수를 사용합니다.

은 다음 코드로 계산할 수 있습니다.

np.dot(np.dot(eigVecs,D),np.linalg.inv(eigVecs))

np.dot(eigVecs,D)는 를 의미하고, np.linalg.inv(eigVecs)은 을 의미합니다.

위 코드의 결과는 원래 행렬인 A 행렬이 되는 것을 알 수 있습니다.

array([[5., 1.],

[3., 3.]])

#### 특이값 분해 (Singular value decomposition)[[60]](#footnote-62)

어떠한 책이나 블로그에서는 LSI를 SVD를 이용한 방식이라고 설명하는 경우도 있습니다. 하지만, 정확하게 말하면 SVD도 고유분해를 사용한 방법이기 때문에 LSI는 결국 고유분해를 근간으로 하는 방법이라고 생각할 수 있습니다.[[61]](#footnote-63) 혼동을 줄이기 위해서 SVD가 무엇인지 설명하도록 하겠습니다. SVD는 정사각행렬이 아닌 사각행렬에 대한 분해 방법이라고 생각하면 됩니다. 다음과 같이 정의됩니다.

여기서 는 정사각행렬이 아닌 사각행렬로 mxn 행렬이라고 표현될 수 있습니다. 는 행렬의 고유벡터를 열로 갖는 행렬이고, V는 의 고유벡터를 열로 갖는 행렬이 됩니다. 그리고 D은 대각행렬인데, 대각원소는 X의 singular values가 되고, 이는 XTX 혹은 XXT의 eigenvalues ()에 루트를 씌운 값 ()이 됩니다 (둘의 값은 같습니다).

SVD를 설명하기 위해 아래와 같은 3x2 행렬을 사용해보도록 하겠습니다.

SVD는 np.linalg.svd() 함수를 가지고 쉽게 계산할 수 있습니다.

X = np.array([[7, 2],

[3, 4],

[5, 3]])

위의 행렬에 대해서 svd() 함수를 아래와 같이 적용할 수 있습니다.

np.linalg.svd(X)

결과는 아래와 같습니다.

(array([[-0.69366543, 0.59343205, -0.40824829],

[-0.4427092 , -0.79833696, -0.40824829],

[-0.56818732, -0.10245245, 0.81649658]]),

array([10.25142677, 2.62835484]),

array([[-0.88033817, -0.47434662],

[ 0.47434662, -0.88033817]]))

위의 결과에서

array([[-0.69366543, 0.59343205, -0.40824829],

[-0.4427092 , -0.79833696, -0.40824829],

[-0.56818732, -0.10245245, 0.81649658]])

가 는 즉, 의 고유벡터를 열로 갖는 행렬입니다. 이는 np.linalg.eig(np.dot(X,X.T))를 사용해서 확인할 수 있습니다. 그리고,

array([[-0.88033817, -0.47434662],

[ 0.47434662, -0.88033817]])

은 에 해당합니다. 가 의 고유벡터를 열로 갖는 행렬입니다. 이는 np.linalg.eig(np.dot(X.T,X))를 이용해서 확인할 수 있습니다.

참고!

고유벡터의 경우는 방향이 서로 반대 방향인지는 중요하지 않습니다. 예를 들어서, 앞에서 살펴본 정사각 행렬인

의 고유벡터 중에서 고유값 6에 해당하는 고유벡터는 array([0.707107, 0.707107]) 이었습니다. 하지만, 방향이 반대인 array([-0.707107, -0.707107])도 해당 고유값의 고유벡터가 될 수 있습니다. 왜냐하면 위의 벡터도 x-y=0이라는 식을 만족하고, 벡터의 길이 = 1인 조건도 만족시키기 때문에 그렇습니다.

#### LSI 이해하기[[62]](#footnote-64)

LSI는 우리가 갖고 있는 텍스트 데이터, 즉 문서의 집합,에 대한 문서-단어행렬 (document-term matrix, DTM)을 이용합니다. DTM에 SVD를 적용하거나 혹은 DTM의 전치행렬과 DTM을 곱해서 나오는 정사각 행렬을 고유분해한 결과를 사용하는 것입니다. 간단한 토이 데이터를 예로 들면서 설명하도록 하겠습니다. 다음과 같은 텍스트 데이터가 있다고 가정합니다.

CORPUS = [

'apple banana apple banana orange',

'apple orange banana orange',

'orange apple apple banana apple',

'carrot spinach eggplant carrot',

'spinach carrot potato spinach',

'carrot potato eggplant eggplant'

]

6개의 문서로 구성된 텍스트 데이터라고 생각할 수 있습니다. LSI를 수행하기 위해서는 일단 전체 텍스트 데이터를 DTM으로 변경해야 합니다. 이를 위해서 sklearn에서 제공되는 CountVectorizer 클래스를 사용합니다 (CountVectorizer는 DTM 행렬의 각 원소 값으로 단어의 빈도 정보를 사용합니다). 텍스트 데이터에서 사용된 전체 단어들, 즉 vocabulary, 은 ['apple', 'banana', 'carrot', 'eggplant', 'orange', 'potato', 'spinach'] 가 됩니다. 그리고 해당 단어들에 대해서 다음과 같은 DTM을 갖습니다.

array([[2, 2, 0, 0, 1, 0, 0],

[1, 1, 0, 0, 2, 0, 0],

[3, 1, 0, 0, 1, 0, 0],

[0, 0, 2, 1, 0, 0, 1],

[0, 0, 1, 0, 0, 1, 2],

[0, 0, 1, 2, 0, 1, 0]], dtype=int64)

위의 DTM을 X라고 표현하도록 하겠습니다. LSI를 하기 위해서는 일단 X의 전치행렬과 X의 곱을 구해야 합니다 (즉, XTX). X가 document-term 행렬이기 때문에 XT는 term-document 행렬이 됩니다. 따라서, XTX는 term-term 행렬이 됩니다. XTX각 원소는 서로 다른 두개의 단어들이 얼마나 자주 혹은 많이 동일한 문서에서 사용되는지를 나타내는 행렬이 됩니다 (즉, co-occurrence 행렬이 되는 것입니다). 우리는 이러한 행렬의 정보를 사용하여 어떠한 단어들이 얼마나 자주 같은 문서에서 사용되는지를 파악하게 됩니다. 그리고 자주 같이 사용되는 단어들이 유사한 주제와 관련이 되어 있다고 간주를 하는 것입니다. 어떠한 단어들이 얼마나 자주 같이 사용되는지에 대한 정보를 고유분해를 사용해서 추출해 낼 수 있는 것입니다.

XTX는 아래와 같이 됩니다. X가 6x7 행렬이므로, XTX는 7x7 행렬이 됩니다 (XTX는 np.dot(X.T,X)를 사용해서 쉽게 구할 수 있습니다).

array([[14, 8, 0, 0, 7, 0, 0],

[ 8, 6, 0, 0, 5, 0, 0],

[ 0, 0, 6, 4, 0, 2, 4],

[ 0, 0, 4, 5, 0, 2, 1],

[ 7, 5, 0, 0, 6, 0, 0],

[ 0, 0, 2, 2, 0, 2, 2],

[ 0, 0, 4, 1, 0, 2, 5]], dtype=int64)

np.linalg.eig()를 사용해서 XTX의 고유값과 고유벡터를 계산합니다. 7x7 행렬이기 때문에 고유값이 7개가 나오고 각 고유값에 대한 고유벡터가 하나씩 계산되게 됩니다. 이러한 고유벡터가 의미하는 것은 XTX에 존재하는 데이터가 흩어진 정도를 설명하는 축이라고 생각할 수 있습니다. 즉, 단어들이 갖고 있는 전체 정보의 정도가 100이라면, 그 100이라는 정보의 양이 7개의 축으로 설명이 되는 것입니다. 그리고, 각 축에 의해 설명이 되어지는 정보의 양은 서로 다릅니다. 그리고 LSI에서는 이러한 하나의 축, 즉 하나의 고유벡터를 하나의 주제 (혹은 주제와 관련된 축)라고 간주합니다. 즉, 우리가 찾을 수 있는 전체 주제는 7개 (이는 vocabulary에 있는 단어의 수와 동일합니다)가 있는 것이고, 그 주제들은 고유벡터를 통해서 표현이 되는 것입니다. 그리고 각 주제에 의해 전체의 코퍼스 데이터가 설명되는 정도가 다른 것입니다. 우리는 보통 전체의 주제들 중에서 데이터를 상대적으로 많이 설명하는 상위 몇개의 주제만을 선택하게 됩니다. 구체적으로 몇개를 선택해야 하는지는 데이터의 특성에 따라 다릅니다.

그리고, 고유값은 각 고유벡터가 데이터를 설명하는 정도를 의미합니다. 예를 들어, 우리가 얻은 고유값이 3개 이고 각 고유값이 6, 3, 1 이라고 한다면 첫번째 고유값에 대한 고유벡터에 의해서 전체 데이터가 설명되는 정도는 전체의 60% (즉, 6/(6+3+1))이 되는 것입니다. 그렇기 때문에 우리는 데이터를 많이 설명하는 몇개의 고유벡터를 선택을 할 때, 각 고유벡터의 고유값의 크기를 가지고 선택할 수 있습니다.

XTX에 대해서 고유벡터와 고유값을 아래와 같이 구해보겠습니다.

eigVals, eigVecs = np.linalg.eig(XTX)

eigVals 에는 7개의 고유값이 저장되어 있습니다.

In: eigVals

Out: array([23.224972, 2.000000, 0.775028, 12.744563, 4.000000, -0.000000, 1.255437])

만약, 우리가 찾고자 하는 주제의 수가 2라고 하는 경우에 우리는 값이 큰 순서대로 2개의 고유값을 선택하고 그에 대한 고유벡터를 선택하면 됩니다. 7개의 고유값중에서 첫번째 (23.224972)와 네번째 값 (12.744563)이 가장 큰 것을 알 수 있습니다. 따라서 우리는 eigVecs에 있는 벡터들 중에서 첫번째와 네번째 고유벡터를 선택합니다.

eigvec1 = eigVecs[:, 0]

eigvec2 = eigVecs[:, 3]

첫번째 고유벡터를 살펴보겠습니다.

In: eigvec1

Out: array([-0.755115, -0.480250, 0.000000, 0.000000, -0.446274, 0.000000, 0.000000])

두번째 주제에 대한 고유벡터를 살펴보도록 하겠습니다.

In: eigvec2

Out: array([0.000000, 0.000000, -0.663535, -0.483527, 0.000000, -0.303519, -0.483527])

각 고유벡터의 원소의 절대값이 의미하는 것은 각 단어와 해당 고유벡터와(즉, 해당 주제와) 관련된 정도입니다.

단어의 순서는 다음과 같습니다.

['apple', 'banana', 'carrot', 'eggplant', 'orange', 'potato', 'spinach']

즉 eigvec1로 나타내어지는 첫번째 주제의 경우는 첫번째 단어인 apple이라는 단어와 관련이 제일 높다는 것을 의미합니다. 그리고 전체적으로 보면 apple, banana, orange 단어들과 관련이 높은 주제라는 것을 알 수 있습니다. 그와 반대로 eigvec2로 나타내어지는 두번째 주제는 세번째 단어인 carrot과 가장 관련이 높은 것으로 나타났습니다. 전체적으로 보면 carrot, eggplant, potato, spinach와 관련이 있는 주제라는 것을 알 수 있습니다. 첫번째 주제가 과일과 관련이 높은 주제이고, 두번째 주제는 채소와 관련이 높은 주제라는 것을 파악할 수 있나요?

그렇다면 각 문서의 주제는 어떻게 찾을 수 있을까요? LSI의 경우는 각 문서의 주제를 각 문서와 각 주제의 유사한 정도로 찾습니다. 이 유사한 정도는 문서를 나타내는 벡터와 주제를 나타내는 고유벡터 간의 내적으로 계산됩니다. 왜냐하면, 내적은 두 벡터의 유사도와 비례하다는 특성을 갖기 때문입니다. 각 문서는 각 단어가 해당 문서에서 사용된 빈도값을 원소로 갖는 벡터로 표현됩니다. 예를 들어 첫번째 문서는 다음과 같은 벡터를 갖습니다.

Doc1 = [2, 2, 0, 0, 1, 0, 0]

위의 벡터는 첫번째 단어인 apple인 문서1에서 두번, 두번째 단어인 banana가 두번, 그리고 orange가 한번 사용되었다는 것을 의미합니다. 그리고 DTM 행렬, 즉 행렬 X의 첫번째 행이 첫번째 문서의 벡터가 되는 것입니다.

첫번째 문서와 첫번째 주제를 나타내는 고유벡터의 유사성은 다음과 같은 내적을 이용해서 계산됩니다.

In: np.dot(doc1,eigvec1) #Doc1•eigvec1

Out: -2.917005078618542

두번째 주제와의 유사성은 아래와 같이 구합니다.

In: np.dot(doc1,eigvec2) # Doc1•eigvec2

Out: 0.0

첫번째 주제와의 유사한 정도는 Doc1•eigvec1 = -2.917005078618542 입니다 (이 값은 -0.755115\*2-0.480250\*2+0\*0+0\*0-0.446274\*1+0\*0+0\*0을 계산해서 나온 값입니다). 다시 말하지만, 이 값의 부호는 중요하지 않습니다. 절대값이 중요합니다.

두번째 주제와의 유사한 정도는 Doc1•eigvec2 = 0이 됩니다. 즉, 두번째 주제와는 아무런 관련이 없다는 것을 의미합니다. 당연히도 그런 것이 두번째 주제를 나타내는 단어들이 문서1에서 하나도 사용되지 않았기 때문입니다.

즉 첫번째 문서는 첫번째 주제와 훨씬 큰 관련이 있는 것을 알 수 있습니다. 이 문서에서는 주제1에 대해서만 다루고 있는 것입니다.

#### gensim을 이용한 LSI[[63]](#footnote-65)

LSI를 이용한 topic modeling은 gensim을 이용해서 할 수 있습니다.[[64]](#footnote-66)

gensim의 경우는 DTM을 만들기 위해서 자체적인 방법을 사용합니다[[65]](#footnote-67). 이를 위해서 각 문서들을 단어들의 리스트로 생성해야 합니다.

docs\_words

[['apple', 'banana', 'apple', 'banana', 'orange'],

['apple', 'orange', 'banana', 'orange'],

['orange', 'apple', 'apple', 'banana', 'apple'],

['carrot', 'spinach', 'eggplant', 'carrot'],

['spinach', 'carrot', 'potato', 'spinach'],

['carrot', 'potato', 'eggplant', 'eggplant']]

corpus, dictionary = build\_doc\_word\_matrix(docs\_words)

LSI 분석을 수행하기 위해서 gensim에서 제공디는 LsiModel 클래스를 사용하는데, 중요하게 지정해야 하는 파라미터가 num\_topics입니다. 여기서는 앞에서의 예와 마찬가지로 이를 num\_topics=2로 세팅합니다.

model = LsiModel(corpus, num\_topics=2, id2word=dictionary)

model.print\_topics() 를 사용하면 각 주제가 어떠한 단어들과 얼마나 연관이 되어 있는지 확인할 수 있습니다.

model.print\_topics()

결과는 아래와 같습니다.

[(0,

'0.755\*"apple" + 0.480\*"banana" + 0.446\*"orange" + -0.000\*"carrot" + -0.000\*"eggplant" + -0.000\*"spinach" + -0.000\*"potato"'),

(1,

'-0.664\*"carrot" + -0.484\*"eggplant" + -0.484\*"spinach" + -0.304\*"potato" + 0.000\*"banana" + -0.000\*"orange" + -0.000\*"apple"')]

앞에서 XTX에 대한 고유벡터를 구한 결과와 동일한 결과가 나옵니다. 다만, 앞에서 구한 고유벡터들과 방향이 반대로 나옵니다. 다시 말하지만, 방향은 중요하지 않습니다.

model.get\_topics()

를 이용해서도 비슷한 결과를 얻을 수 있습니다.

각 문서가 각 주제와 얼마나 관련이 있는지를 확인하기 위해서는 다음 코드를 사용합니다.

model[corpus[0]] # corpus[0]는 첫번째 문서를 의미합니다.

[(0, 2.917005078618542)] #첫번째 문서가 첫번째 주제와 갖는 관련도 정도입니다.

첫번째 주제와의 내적값이 나옵니다. 앞에서 구한 값과 절대값이 같은 것을 알 수 있습니다. 이 값의 절대값이 크면 클수록 해당 주제에 대해서 많이 다루고 있는 것입니다. 지금은 두번째 주제에 대한 내적값이 0이라고 그 값이 출력되지 않았습니다.

**실제 데이터를 사용한 LSI 분석 코드[[66]](#footnote-68)**

여기서는 군집화 분석에서 사용된 15개의 신문기사로 구성된 텍스트 데이터를 사용해서 LSI topic modeling을 수행해 보도록 하겠습니다. 전처리를 하고, build\_doc\_word\_matrix()을 이용해서 DTM을 만드는 것은 앞의 내용과 동일하기 때문에 여기서는 생략하도록 하겠습니다. 우리가 다루는 텍스트 데이터의 신문기사들이 다루는 주제의 수는 5라고 했습니다. 따라서 여기서 찾고자 하는 주제의 수를 5로 설정합니다.

model = LsiModel(corpus, num\_topics=5, id2word=dictionary)

각 주제가 어떠한 단어들과 관련이 높은지 확인해 보겠습니다. 여기서는 관련이 높은 상위 10개의 단어들만을 사용합니다.

In: model.print\_topics(num\_words=10)

Out: [(0,

'0.451\*"trump" + 0.421\*"president" + 0.256\*"mccain" + 0.178\*"mccains" + 0.144\*"john" + 0.125\*"admission" + 0.121\*"week" + 0.119\*"senator" + 0.117\*"house" + 0.113\*"trade"'),

(1,

'-0.451\*"admission" + -0.353\*"student" + -0.297\*"harvard" + -0.190\*"college" + -0.189\*"applicant" + -0.180\*"university" + 0.177\*"trump" + -0.168\*"case" + -0.161\*"race" + -0.150\*"court"'),

(2,

'-0.281\*"trade" + -0.277\*"economy" + -0.245\*"export" + -0.244\*"china" + -0.213\*"september" + -0.190\*"survey" + -0.188\*"order" + -0.175\*"growth" + -0.167\*"month" + -0.167\*"company"'),

(3,

'0.295\*"jacksonville" + 0.251\*"people" + 0.233\*"tournament" + 0.232\*"williams" + 0.178\*"game" + 0.163\*"victim" + 0.149\*"video" + 0.133\*"hospital" + 0.133\*"area" + 0.132\*"landing"'),

(4,

'0.367\*"agreement" + 0.319\*"mexico" + 0.300\*"canada" + 0.281\*"deal" + 0.187\*"negotiation" + 0.185\*"trade" + 0.166\*"issue" + 0.158\*"talk" + -0.155\*"mccain" + 0.145\*"official"')]

위의 결과를 보면, 첫번째 주제는 trump, president, mccain이라고 하는 단어들과 상대적으로 관련이 높은 주제인 것을 알 수 있습니다. 네번째 주제의 경우는 'jacksonville', 'people' 등과 관련이 높은 것을 알 수 있습니다.

각 문서는 어떠한 주제와 관련이 높은지 확인해 보겠습니다. 아래는 첫번째 문서에 대한 결과입니다. 절대값이 큰 주제가 관련이 제일 높은 주제입니다. 주제 3 (네번째 주제)에 대한 값이 -17.6정도로 절대값이 가장 큰 것을 알 수 있습니다. 즉, 첫번째 문서는 네번째 주제와 관련이 제일 높은 것입니다. 위의 결과를 확인해 보면 네번째 주제는 'jacksonville', 'people', 'tournament', 'williams' 등의 단어와 관련이 높은 주제입니다.

In: model[corpus[0]]

Out: [(0, 4.925546230897631),

(1, -1.9153485857694872),

(2, -2.602220509917118),

(3, -17.62942811421356),

(4, 1.6130649698920996)]

첫번째 신문기사의 내용을 살펴보겠습니다. 해당 신문기사를 보면 Jacksonville, tournament, game 등의 단어들이 자주 등장하는 것을 확인 할 수 있습니다.

두번째 신문기사에 대한 결과를 확인해 보겠습니다. 첫번째 신문기사와 마찬가지로 네번째 주제와의 관계가 가장 큰 것으로 나왔습니다. 두 신문기사는 동일한 주제를 다루고 있기 때문에 결과가 잘 나온 것으로 해석할 수 있습니다.

In: model[corpus[1]]

Out: [(0, 6.20864805708728),

(1, -1.375952983106561),

(2, -1.9458559231975605),

(3, -18.79377386579195),

(4, 1.3841527671638543)]

나머지 신문기사에 대해서도 여러분들이 직접 한번 확인해 보길 바랍니다.

## LDA (Latent Dirichlet Allocation)

이번 섹션에서는 사용빈도가 높은 확률 기반의 토픽 모델링 방법인 LDA에 대해서 살펴보겠습니다. LDA를 이해하기 위해서는 기본적으로 LDA에서 사용되는 확률 분포들에 대해서 알고 있을 필요가 있습니다. 이를 위해서 LDA에서 중요한 역할을 하는 디리클레(Dirichlet) 확률 분포와 다항(Multinomial) 확률 분포에 대해서 먼저 알아보겠습니다.

### 디리클레 분포

디리클레 분포는 베타 (Beta) 분포의 일반화된 형태이기 때문에, 이해를 쉽게하기 위해 베타 분포를 먼저 살펴보겠습니다.

#### 베타 분포

베타 분포는 0~1 사이의 값을 갖는 **하나의 연속 변수**에 대한 확률 분포입니다(여기서는 해당 연속 변수를 X라고 표현하겠습니다). 그리고 확률 분포의 형태가 두 개의 파라미터에 의해서 결정됩니다. 보통 α, β라고 표현됩니다. 확률 분포의 확률밀도함수(probability density function)는 아래와 같습니다.

,

where

여기서 는 감마 함수입니다. 감마 함수는 아래와 같습니다.

그리고 같은 특성을 갖습니다.

베타 분포는 α, β의 값에 따라서 다음과 같은 다양한 형태를 갖습니다.

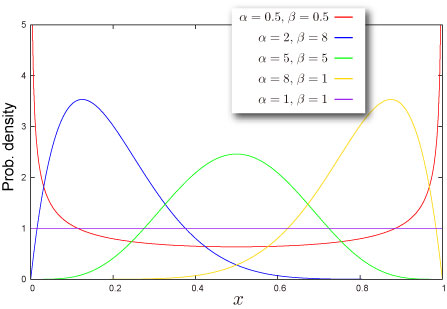


그림 . 베타 분포의 형태

그림 1.1에서 볼수 있는 것 처럼 α와 β값에 따라서 X 변수가 1에 가까운 값들을 취할 확률이 더 큰지, 반대로 0에 가까운 값들을 취할 확률이 더 큰지가 결정됩니다. α값이 β값보다 상대적으로 더 큰 경우에는 변수가 1에 가까운 값을 취할 확률이 더 커집니다(노란색 선). 반대로 β값이 α값 보다 상대적으로 더 큰 경우에는 변수가 0에 가까운 값을 가질 확률이 더 커집니다 (파란색 선). 디리클레 분포에서도 분포의 형태를 결정하는 파라미터들이 이러한 역할을 합니다. α와 β값이 동일한 경우에는 좌우가 대칭인 분포의 형태가 됩니다. α = β = 1 인 경우에는 균등(Uniform) 분포, U(0, 1)과 같은 형태가 됩니다. α와 β값 모두 증가할 수록 X=0.5일 확률이 증가하는 대칭인 분포가 됩니다.

#### 디리클레 분포

디리클레 분포는 베타 분포를 일반화한 것이라고 생각할 수 있습니다. 즉, [0,1] 사이의 값을 취하는 여러 개 (2개 이상)의 연속 변수에 대한 확률 분포가 되는 것입니다. 예를 들어 0 ~ 1 사이의 값을 갖는 k 개의 연속변수가 있다고 가정합니다. 이러한 변수들을 라고 표현하면, 디리클레 분포는 에 대한 확률 분포, 즉가 되는 것입니다. 는 0과 1 사이의 값을 갖고 각 변수가 취하는 값의 합은 1이 되어야 합니다. 즉,

을 만족해야 합니다.

디리클레 분포의 확률밀도함수는 아래와 같습니다.

where

where

디리클레 분포의 형태는 파라미터 벡터인 **α**에 의해 결정됩니다. **α** 벡터는 원소의 갯수가 k개인 벡터이고 (변수의 수와 동일합니다), 각 원소가 해당 디리클레 분포의 형태를 결정하는 파라미터가 됩니다. 라고 표현될 수 있습니다. 여기에서 는 베타 분포에서의 α, β와 유사한 역할을 한다고 생각할 수 있습니다. 디리클레 분포에서는 의 값이 커질수록 확률분포에서 갖는 변수의 비중이 더 커지게 됩니다. 그리고, 인 경우에는 각 방향이 서로 대칭인 확률 분포를 갖게 됩니다. 그 값들이 클수록 가운데가 볼록한 형태가 됩니다 (그림 1.2 참고). 그림 1.2는 변수가 3개인 경우를 보여주고 있습니다.

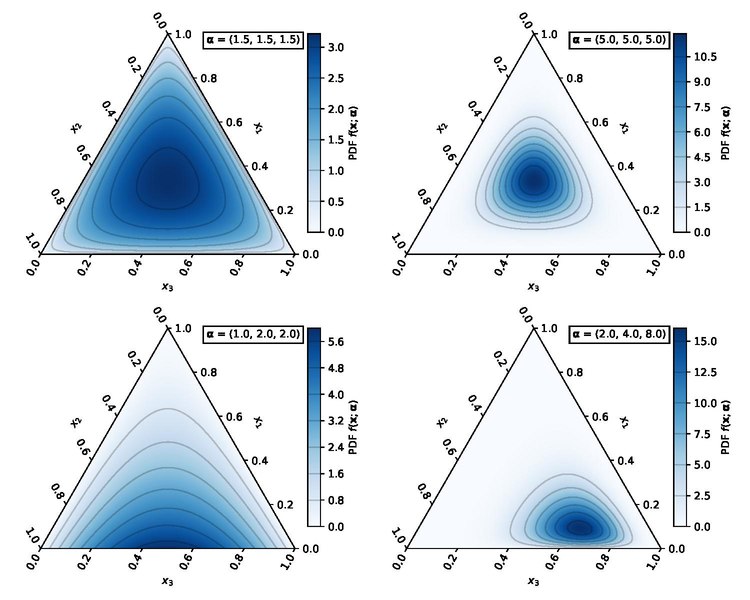


그림 . 디리클레 분포의 형태

### 다항 (Multinomial) 분포

LDA를 이해하기 위해서 알아야 하는 분포가 하나 더 있습니다. 바로 다항 분포입니다. 다항 분포는 이항 (Binomial) 분포의 일반화 형태이기 때문에 이해를 쉽게 하기 위해서 이항 분포에 대해서 먼저 간략하게 살펴보겠습니다.

#### 이항 분포

‘성공’과 ‘실패’ 둘 중 하나의 값이 결과로 나올 수 있는 시행을 n 번 수행할 때, 나오는 ’성공’의 횟수를 값으로 취하는 변수를 X라고 하는 경우, 이 X 변수는 이항 분포를 따릅니다. 그리고 보통 ‘성공’을 1로, ‘실패’를 0으로 표현합니다. 한 번 시행했을 때 1의 값이 나올 확률을 p라고 하는 경우, 이항 분포의 형태를 결정하는 파라미터는 n과 p가 됩니다. 여기서 p는 0~1사이의 값을 갖는 확률이므로 베타 분포를 따르는 또 다른 변수라고 생각할 수 있습니다.

이항 분포를 갖는 대표적인 예가, 동전을 n번 던졌을 때 나오는 앞면의 수를 취하는 변수가 됩니다. 여기에서 동전을 한번 던졌을 때 나올 수 있는 값은 앞면과 뒷면 두개가 되고, 앞면을 1 또는 ‘성공’으로 뒷면을 0 또는 ‘실패’라고 간주할 수 있습니다. 그리고 한 번 던졌을 때 앞면이 나올 확률이 p가 되는 것입니다.

#### 다항 분포

다항 분포는 이항 분포의 일반화된 분포입니다. 이항 분포의 경우, 한 번의 시행에서 나올 수 있는 값이 오직 두 개 (즉, ‘성공’과 ‘실패’)인 반면, 다항 분포의 경우는 한 번의 시행에서 나올 수 있는 값이 여러 개(즉, 3개 이상)가 됩니다. 주사위를 던지는 경우가 그러한 예입니다. 주사위를 한 번 던지는 경우 나올 수 있는 값의 수는 6개가 됩니다. 그리고 각 값이 나올 확률은 (주사위가 fair하다고 가정한다면) 1/6이 됩니다. 여러 개의 결과값을 가질 수 있는 시행을 n 번 수행했을 때 각 결과값이 나오는 횟수를 값으로 취하는 변수들의 분포를 다항 분포라고 합니다. 주사위를 던지는 경우의 예에서는 변수가 6개가 존재하는 것입니다. 이를 라고 표현하겠습니다. 주사위를 n번 던졌을 때 1의 값이 나오는 횟수를 결과로 취하는 변수가 , 2의 값이 나오는 횟수를 결과로 취하는 변수가 가 되는 것입니다. 각 변수가 취하는 값을 라고 하는 경우에 다음과 같은 방정식이 만족하게 됩니다.

한 번 시행을 했을 때 K개의 결과가 나올 수 있고, j번째 결과가 나올 확률을 라고 하는 경우에 다음 방정식을 만족해야 합니다.

주사위 예의 경우는 K의 값이 6이 되고, 아래 방정식이 만족해야 합니다.

다항 분포의 형태를 결정하는 파라미터에는 n과 가 있습니다. 여기에서 가 디리클레 분포를 따르는 또 다른 변수라고 생각할 수 있습니다. 따라서, 디리클레 분포와 다항 분포가 크게 관련이 있는 것입니다. 베이지안 추론에서는 디리클레 분포를 다항 분포의 켤레 사전 (conjugate prior) 분포라고 하는데, LDA에서는 두 분포가 갖는 이러한 관계를 이용합니다.

참고로 다항 분포에서 시행이 1인 경우 (즉, n=1인 경우)의 분포를 카테고리 (Categorical) 분포라고 합니다.

### LDA의 이해

지금까지 LDA 모형을 이해는데 필요한 기본적인 확률 분포 개념에 대해서 공부했습니다. 이를 토대로 이제 LDA 모형이 어떻게 작동되는지 설명해 보도록 하겠습니다. LDA는 확률적 생성 모형입니다. 즉, 우리가 관측하는 혹은 수집한 텍스트 데이터에 존재하는 문서가 특정한 확률 모형 즉, 확률 분포를 통해서 생성되었다라고 가정합니다. LDA는 이러한 확률 분포로 서로 다른 2 종류의 확률 분포를 사용합니다. 하나는 문서별 주제 분포이고, 다른 하나는 주제별 단어 분포가 됩니다. 둘 모두 디리클레 분포입니다.

#### LDA에서 사용되는 확률 분포

1) 분포1: 문서별 주제 분포

그 첫 번째 확률 분포는 문서별 주제 분포가 됩니다. 전체 주제의 수가 K라고 하는 경우, 특정 주제 k가 특정 문서 d에서 다뤄질 확률은 로 표기됩니다. 이는 문서 d에서 주제 k가 다뤄지는 정도를 의미하게 됩니다. 해당 주제가 해당 문서에서 다뤄지는 비중이 클 수록 해당 주제의 확률값이 크게 됩니다. 그리고

이 됩니다.

2) 분포2: 주제별 단어 분포

두 번째 종류의 확률 분포는 주제별 단어 분포가 됩니다. 이 분포는 각 단어가 특정한 주제와 얼마나 관련이 있는지를 나타냅니다. 특정 단어 j가 특정 주제 k와 관련이 있는 정도를 라고 표현할 수 있습니다. 단어가 갖는 확률이 높을수록 그 주제와 관련이 높은 단어입니다. 그리고 말뭉치 데이터에서 사용된 총 단어의 수가 N개인 경우 특정 주제 k에 대해서 다음과 같은 식이 만족합니다.

LDA는 이 두 가지 종류의 확률 분포를 통해서 **특정 문서에서 사용되는 각 단어가 선택되어 졌다고 (혹은 생성되었다고) 가정**합니다. 그래서 LDA를 확률적 생성 모형이라고 합니다. 그리고 우리가 가지고 있는 텍스트 데이터의 각 문서에 출현한 단어들의 횟수 정보를 사용하여 역으로 우리가 가지고 있는 문서 데이터를 가장 잘 설명하는 이 두 종류의 분포의 구체적인 형태를 추정하게 됩니다. 즉, 문서별 주제 분포와 주제별 단어 분포의 구체적인 확률값을 추정하게 되는 것입니다.

LDA에서는 다음과 같은 순서를 통해서 문서 d에서 특정 단어 (단어 n라고 표현하겠습니다)가 선택되어 사용되었다고 가정합니다.[[67]](#footnote-69) [[68]](#footnote-70) 이러한 과정을 좀 더 구체적으로 살펴보도록 하겠습니다.

#### 특정 단어 (단어 n) 생성 순서

1) 문서 d에 대한 주제 분포, ,를 선택한다. 는 디리클레 분포를 따른다. 즉,

\* 여기서 는 각 주제가 문서 d에서 다뤄지는 비중에 대한 확률값들을 원소로 갖는 벡터라고 생각할 수 있습니다.

\*\* 는 해당 디리클레 분포의 형태를 결정하는 파라미터 벡터로, 원소의 수는 주제의 수와 같습니다. 즉,

2) 위의 주제 분포에 대해서 단어 n은 다음과 같은 순서를 통해서 선택되어 진다.

2.1) 주제 할당 (topic assignment): 단어 n의 주제를 선택

단어 n의 주제를 할당하기 위해 LDA 모형은 잠재변수를 하나 사용합니다. 해당 잠재변수는 으로 표기됩니다. 해당 잠재변수는 카테고리 분포를 따릅니다. 즉, 아래와 같이 표현됩니다.

만약 K개의 주제가 있다라고 한다면 이 취할 수 있는 값은 0, 1, …, K-1이 되는 것이고, 각 값을 취할 확률이 , 즉, 문서 d의 주제 분포에 의해서 결정된다는 것입니다. 만약, 여기에서 가 선택되어 진다면, 단어 n의 주제는 j (즉, Topic j)가 되는 것입니다.

2.2) 해당 주제에 대한 주제별 단어 분포 선택

단어 n의 주제가 선택되어진 후에는 해당 주제에 대한 주제별 단어분포를 선택합니다. 만약, 주제 j (즉, )가 선택된 경우에는 주제 j에 대한 단어 분포를 선택합니다. 이 분포는 텍스트 데이터에 존재하는 각 단어들이 주제 j와 관련된 정도를 나타내는 확률 분포라고 생각할 수 있습니다. 이 주제별 단어 분포를 라고 표현하겠습니다. 이는 또 다른 디리클레 분포가 됩니다. 아래와 같이 표현됩니다.

여기서 는 해당 디리클레 분포의 형태를 결정하는 파라미터 벡터로, 원소의 수는 전체 단어의 수와 같습니다. 만약, 코퍼스에서 사용된 전체 단어의 수가 N이라면, 이 됩니다.

2.3) 해당 주제별 단어 분포로부터 단어 n 을 선택

마지막으로 또 다른 카테고리 분포에서 단어 n을 선택합니다. 이 카테고리 분포의 파라미터는 가 됩니다. 즉,

보통 문헌에서는 에서 단어 n이 선택되어질 확률을 와 같은 조건부 확률로 나타냅니다. 이는 특정 단어(단어 n)가 선택되어질 확률이 해당 단어의 주제 (즉, )와 해당 주제의 단어분포 (즉, )에 의해서 영향을 받는다는 것을 말합니다.

블레이 논문 (Blei, 2012) 에서는 이러한 단어 생성 과정을 아래와 같은 그래프 모형으로 표현하기도 했습니다.



그림 . 단어 생성 과정

여기에서 N는 텍스트 데이터에 있는 전체 단어의 수이며, D는 전체 문서의 수, 그리고 K는 전체 주제의 수가 됩니다.

#### 분포의 형태 파악하기

LDA 모형은 이러한 과정을 거쳐 문서 데이터가 생성되었다라고 가정하고, 역으로 우리가 가지고 있는 문서 데이터를 가장 잘 설명하는 두 종류의 분포를 찾습니다. 즉, (문서별 주제 분포)와 (주제별 단어 분포)를 찾게 되는 것입니다. 이를 찾기 위해 LDA 모형은 베이지안 추론 방법을 사용합니다. 베이지안 추론 방법에 대한 자세한 설명은 본 책의 범위를 벗어납니다. 하지만, 베이지안 추론 방법을 이해하지 않고 LDA를 이해하는 것이 거의 불가능하기 때문에 본 책에서는 베이지안 추론 방법에 대해서 간략하게 설명하도록 하겠습니다.

1) 베이지안 추론 (Bayesian inference)

베이지안 추론은 우리가 가지고 있는 데이터를 근거로 하여 해당 데이터를 생산하는데 사용된 확률 분포의 파라미터의 값을 추론하는 방법을 말합니다. 이를 위해서 베이즈 규칙 (Bayes' rule)을 사용합니다 (베이즈 규칙은 경우에 따라서 Bayes' theorem이라고도 합니다).

베이즈 규칙은 아래와 같이 조건부 확률을 기반으로 하고 있습니다. 가 의미하는 것은 y가 발생했다는 조건에서 가 발생할 확률을 의미합니다.

여기서 는 우리가 궁금해 하는 파라미터가 되고 y는 우리가 현재 가지고 있는 (에 영향을 받아 생성된) 데이터를 의미합니다. : 의 사후분포(Posterior distribution), : ( 에 영향을 받는) 우리가 가지고 있는 데이터의 확률 (즉, Likelihood), : 의 사전확률 (Prior distribution), : 데이터의 확률 (증거, evidence) 라고 합니다. 는 가 확률분포가 될 수 있도록 전체의 합 = 1이 되도록 만들어주는 역할을 합니다 (이를 normalizing factor라고 합니다). 가 연속변수인 경우에는 가 됩니다.

LDA에서 우리가 알고자하는 확률 분포 와 가 됩니다. 이들의 구체적인 형태를 추론하기 위해서 현재 우리가 가지고 있는 데이터 (W라고 표현하겠습니다)를 사용합니다. 베이지안 추론 방법을 사용하면 아래와 같이 표현됩니다.

즉, 위의 식을 이용해서 W라는 데이터를 토대로 역으로 와 의 형태를 추정하는 것입니다.

는 와 같이 표현될 수 있고, 로 표현될 수 있습니다. 하지만, 실제 계산에서 를 구하는 것이 쉽지 않습니다. 를 직접 구하지 않고, 의 형태를 파악하기 위해서 MCMC (Markov Chain Monte Carlo) 시뮬레이션 방법이 사용됩니다. 그 중에서도 깁스 샘플링(Gibbs sampling)[[69]](#footnote-71)이라는 방법이 사용됩니다 (Blei, 2012).

위의 베이지안 추론을 통해 와 의 구체적 형태를 파악하기 위해서는 먼저 와 분포의 파라미터들, 즉, ~와 ~ 에서의 벡터와 벡터의 값을 먼저 결정해야 합니다. 이 값들은 결정하는 방법에는 여러가지가 있으나, 원 논문 (Blei et al., 2003)에서는 최대우도추정(maximum likelihood estimation) 방법을 사용했습니다.[[70]](#footnote-72) 이는 우리가 가지고 있는 데이터의 확률을 최대화하는 벡터와 벡터값을 찾는 방법입니다. 와 의 값으로 표현되는 우리가 가지고 있는 전체 문서데이터, 즉, W의 확률은 로 표현됩니다. 즉, 의 값을 최대로 하는 와 값을 선택하는 것입니다.

### Gensim을 이용한 LDA 수행하기

본 섹션에서는 gensim 모듈을 이용해서 LDA를 수행하는 방법에 대해서 알아보도록 하겠습니다. [[71]](#footnote-73)

gensim을 이용한 LDA 분석은 아래와 같이 LdaModel이라고 하는 클래스를 사용해서 수행합니다. 해당 클래스의 생성자 함수를 보면 주요한 파라미터가 5개가 있습니다. 첫 번째는 분석하고자 하는 텍스트데이터에 대한 것이고 (아래에서 첫번째 인자로 제공됩니다), 두 번째가 해당 텍스트 데이터에서 찾고자 하는 주제의 수 (num\_topics), 그리고 단어들의 ID 정보를 입력 받는 id2word가 됩니다.

lda\_model = models.ldamodel.LdaModel(corpus, num\_topics=NUM\_TOPICS,

id2word=dictionary, alpha='auto', eta='auto')

위의 첫번째 인자로 제공된 corpus는 전체 데이터에 대한 DTM이라고 생각할 수 있습니다. corpus와 id2word 파라미터의 인자인 dictionary는 LSI 분석에서 사용했던 build\_doc\_word\_matrix() 함수를 이용해서 얻을 수 있습니다.

그 다음으로 설정해야 하는 것이 alpha와 eta입니다. alpha는 문서-주제 디리클레 사전 분포의 파라미터를 의미하며, eta는 주제-단어 디리클레 사전 분포의 파라미터입니다. alpha와 eta가 취할 수 있는 값은 ‘symmetric’, ‘asymmetric’, ‘auto’가 있습니다. 이중에서 보통은 ‘symmetric’ 이나 ‘auto’가 사용됩니다. ‘symmetric’은 사전 분포가 갖는 파라미터의 값들을 동일하게 하는 것입니다. 문서-주제 분포의 경우, 특정 주제에 미리 더 많은 가중치를 주지 않겠다, 즉, 우리가 가지고 있는 데이터의 정보를 반영하기 이전에 존재하는 사전 분포는 주제들의 비중이 동일한 분포를 사용하겠다는 것입니다. 그리고 주제-단어 분포의 경우에는 단어들의 비중이 동일한 사전 분포를 사용하겠다는 것을 의미합니다. ‘auto’는 데이터의 내용을 반영하여 사전 분포의 파라미터값을 결정한다는 것을 의미합니다.

‘symmetric’ 대신 특정한 숫자값을 사용하여 symmetric 한 사전 분포를 사용할 수 있습니다. 디리클레 분포 부분에서 설명한 것 처럼 파라미터의 값들이 클수록 가운데가 볼록한 분포가 됩니다. 그리고 각 파라미터에 대한 값을 별도로 지정하는 1차원 array를 사용할 수 있습니다.

위의 코드를 수행하게 되면 LDA를 이용한 토픽 모델링의 결과가 lda\_model이라고 하는 객체에 저장됩니다.

#### 주제별 단어 분포 보기

lda\_model에 대해서 주제별 단어 분포 결과를 보기 위해서는 print\_topic\_words() 함수를 사용합니다. print\_topic\_words()는 주제별 단어 분포를 확인하기 위한 사용자 정의 함수이며, 아래와 같이 정의되어 있습니다.

def print\_topic\_words(model):

for topic\_id in range(model.num\_topics):

word\_probs = model.show\_topic(topic\_id, NUM\_TOPIC\_WORDS)

print("Topic ID: {}".format(topic\_id))

f.write(str(topic\_id)+'\n')

for word, prob in word\_probs:

print("\t{}\t{}".format(word, prob))

f.write(str(word)+'\t'+str(prob)+'\n')

print("\n")

f.close()

print\_topic\_words(lda\_model)와 같이 입력하면 그 결과는 아래와 같습니다. 각 주제에 대해서 관련이 높은 15개의 단어들만 결과로 출력되었습니다. 출력되는 단어의 수는 여러분이 해당 파이썬 코드에서 변경할 수 있습니다.

첫 번째 주제 (Topic 0)와 관련이 높은 단어들은 '국민의당', '수석', '검찰' 등인 것을 알 수 있습니다 (여러분들이 얻은 결과는 본 책의 것과 다를 수 있습니다). 여러분이 결과를 해석할 때 주의해야 하는 것이 무엇이냐면, LDA 분석의 결과는 각 주제의 이름을 제공하지 않는다는 것입니다. 각 주제가 무엇에 대한 것인지는 사용자가 해당 주제와 관련이 높은 단어들을 참고하여 판단해야 합니다. 예를 들어, 두 번째 주제의 경우 학생, 취업 등이 해당 주제가 될 수 있을 것 같습니다.

Topic ID: 0

국민의당 0.02494213543832302

수석 0.01919584721326828

검찰 0.014740750193595886

의혹 0.014311060309410095

리베이트 0.0092662014067173

위원장 0.009015046991407871

홍보 0.008896920830011368

총선 0.008719006553292274

새누리당 0.008626600727438927

선거 0.0079408073797822

원내 0.00715244747698307

상임 0.006100842729210854

청와대 0.006053857039660215

김수민 0.005916978232562542

원내대표 0.005769731942564249

Topic ID: 1

학생 0.011346916668117046

학교 0.009760268963873386

취업 0.0083623630926013

청년 0.007238992024213076

교육 0.006863405928015709

고용 0.00662551075220108

근로자 0.0057957107201218605

대기업 0.00546283507719636

지원 0.005437888205051422

임금 0.0054346779361367226

공무원 0.005081988871097565

대학 0.005057626869529486

여성 0.0048692477867007256

일자리 0.004667287692427635

채용 0.0045392513275146484

(이하 생략)

#### 문서별 주제 분포 보기

이번에는 문서별 주제 분포를 확인해 보겠습니다. 이는 LdaModel에서 제공되는 get\_document\_topics() 함수를 이용해서 확인할 수 있습니다. 예를 들어서, 첫번째 문서 (즉, corpus[0])의 주제 분포를 보기 위해서 다음과 같이 입력합니다.

lda\_model.get\_document\_topics(corpus[0], minimum\_probability=0.1)

minimum\_probability 파라미터의 값을 0.1로 설정하면, 0.1보다 확률이 낮은 주제에 대한 결과는 출력되지 않습니다.

실행 결과는 다음과 같습니다.

[(1, 0.5783152), (19, 0.2130413)]

해당 문서에서 다뤄진 정도가 0.1 보다 큰 주제 2개만 결과로 출력되었습니다. 주제의 ID가 1인 주제가 0.58 정도로 가장 큰 값을 가지고 있습니다. 이는 해당 주제가 해당 문서에서 가장 많이 다뤄졌다는 것을 의미하고(58% 정도), 이는 해당 주제와 관련된 단어들이 그 만큼 해당 문서에서 많이 사용되었다는 것을 의미합니다. 그렇다면 주제 ID가 1인 주제가 무엇일까요? 이를 위해서 앞에서 살펴본 주제별 단어 분포 결과를 확인해 볼 수 있습니다. 해당 주제와 관련이 높은 단어들로 '학생', '취업' 등이 있었습니다. 첫번째 신문기사의 제목을 확인해 보면 " 대기업 연봉 6천700만원…中企보다 2천500만원 더 …" 인 것을 알 수 있습니다.

#### 주제 수 결정하기

LDA를 수행할 때 중요하게 결정해야 하는 것이 주어진 텍스트 데이터에서 찾고자 하는 전체 주제의 수가 됩니다. 주제의 수는 사용자가 직접 결정하는 하이퍼파라미터가 되는데, 토픽의 수를 결정할 때 크게 두 가지 정보를 사용할 수 있습니다. 하나가 혼잡도(Perplexity)이고, 다른 하나가 응집도(Coherence)[[72]](#footnote-74) 값이 됩니다.

##### 1) 혼잡도(Perplexity)

혼잡도는 데이터의 확률 (조금 더 정확하게는 코퍼스에 존재하는 단어들의 확률이 됩니다), 즉 데이터에 대한 우도(Likelihood)와 관련이 있는 개념입니다. 데이터의 확률과 혼잡도는 반비례 관계를 갖습니다. 다음과 같이 표현될 수 있습니다.

여기에서 가 데이터의 확률인 우도가 됩니다. 그리고 데이터의 확률은 우리의 모형이 우리가 가지고 있는 데이터를 잘 설명할수록 그 값이 커지게 됩니다. 즉 해당 데이터가 우리가 가지고 있는 모형에 의해서 생성되었을 확률이 그 만큼 크다는 것을 의미합니다. 따라서 우리는 여러가지 결과들 중에서 데이터의 확률 즉, 우도값을 크게하는 결과를 선택해야 하는 것입니다. 이는 혼잡도의 값이 낮은 경우의 주제 수를 사용해야한다는 것을 의미합니다. 예를 들어 보도록 하겠습니다. 값이 0.1 인 경우와 0.2인 경우가 있다라고 가정합니다. 는 모형의 설명력을 나타내기 때문에 0.2인 경우의 모형의 설명력이 더 좋다는 것을 의미합니다. 즉, 우리는 0.1인 경우의 모형이 아닌 0.2에 해당하는 모형을 선택해야 합니다 (그에 해당하는 주제의 수를 선택해야 하는 것입니다). 인 경우에는 이 되고, 인 경우에는 이 됩니다. 즉, 의 값이 모형의 설명력이 더 좋은 경우(즉, 인 경우)에 더 작은 것입니다. gensim의 LdaModel에서는 log\_perplexity() 함수를 사용하여 log perplexity값을 사용합니다. 위의 값들에 대해서 을 구해보겠습니다. 에 대해서는 , 그리고 에 대해서는 가 됩니다. 따라서 우리는 log perplexity값이 작은 경우의 모형을 선택해야 하는 것입니다. 이는 log perplexity값이 가장 작은 경우의 주제의 수가 혼잡도를 기반으로 했을 때 가장 적합하다는 것을 의미합니다.

##### 2) 응집도 (Coherence)

응집도는 각 주제와 관련이 높은 상위 k개의 단어들 간의 유사도를 의미합니다. 단어 간의 유사도는 PMI (Pointwise Mutual Information)를 이용합니다. 두개의 단어에 대해서 PMI는 아래와 같이 정의됩니다.

는 단어 와 단어 가 얼마나 같이 자주 사용되는지를 의미합니다. 즉, 두 단어가 같이 자주 사용될 수록 값이 커지게 되고, 더불어 값도 커지게 됩니다. 같은 주제와 관련이 높은 단어들의 경우는 PMI값이 커야하기 때문에 우리는 보통 응집도 값을 제일 높게하는 주제의 수를 선택하게 됩니다.

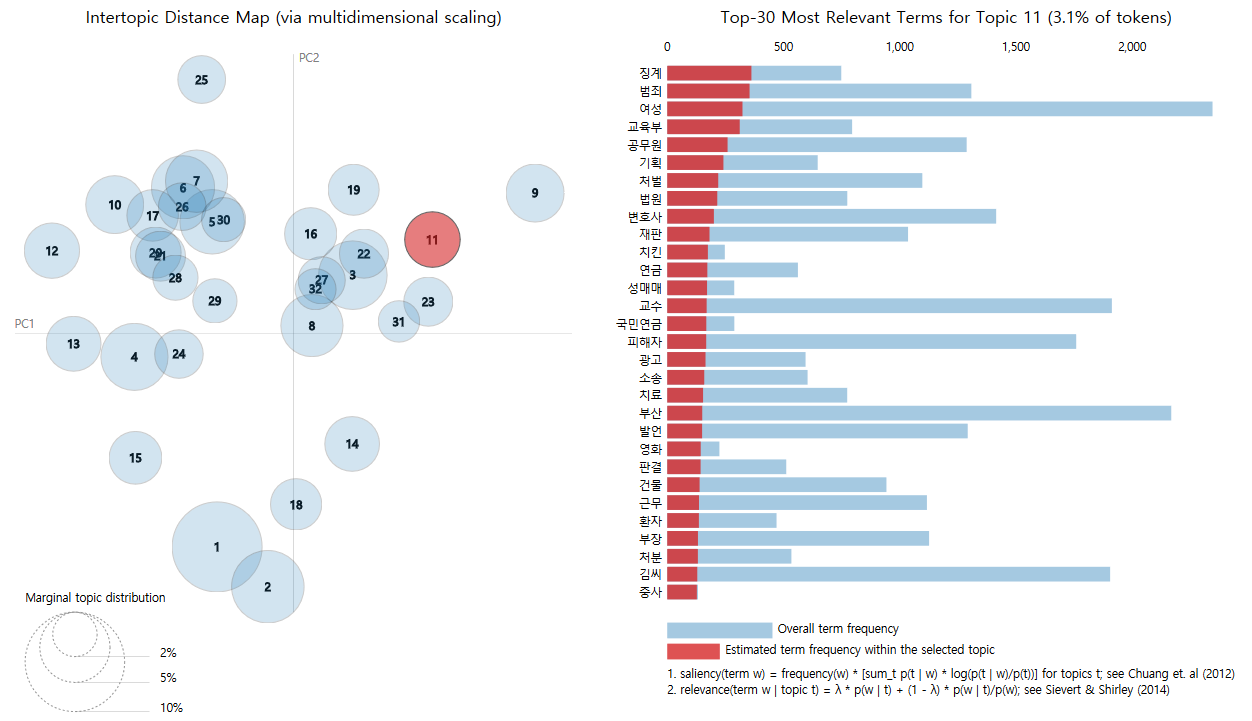
###박스 시작###

주의! 하지만, 혼잡도나 응집도에 의해 제안되는 주제의 수가 항상 정확한 것은 아닙니다. 오히려 많은 경우에 정확하지 않을 수도 있습니다. 이러한 경우에는 텍스트 데이터에 대한 지식이나 관련 도메인 지식을 활용해서 적절한 주제의 수를 선택해야 합니다.

###박스 끝###

#### LDA 결과 시각화

LDA 분석 결과를 시각화하기 위해서 pyLDAvis라는 모듈을 사용할 수 있습니다. 해당 모듈은 윈도우의 경우 명령 프롬프트 창에 pip install pyldavis를 입력해서 설치합니다. 이와 관련된 코드는 LDA\_example.ipynb 파일을 참고하세요. 예제 데이터에 대한 시각화 결과는 아래와 같습니다.



왼쪽의 그림은 32개의 주제가 2차원 공간에 어떻게 위치하는지를 보여주고 있고, 그리고 오른쪽의 그림은 특정 주제 (위의 경우는 주제 11)와 관련이 높은 단어들의 빈도를 보여주고 있습니다.

1. 기계학습에서는 피쳐 (feature)라고도 합니다. [↑](#footnote-ref-2)
2. 범주형 변수는 명목 변수 (nominal variable)라고도 합니다. [↑](#footnote-ref-3)
3. MAE (mean absolute errors)와 같은 다른 비용함수도 있습니다. [↑](#footnote-ref-4)
4. 뉴튼-랩슨 (Newton-Raphson) 방법 등의 다른 방법도 있습니다. [↑](#footnote-ref-5)
5. 아래로 볼록한 형태의 함수를 볼록 함수 (convex function)이라고 합니다. [↑](#footnote-ref-6)
6. 의 값은 사용자가 결정하게 되는데 보통 0 – 1 사이의 값을 취합니다. 이렇게 사용자가 그 값을 직접 결정해야 하는 것을 하이퍼파라미터라고 합니다. [↑](#footnote-ref-7)
7. 이러한 차이 이외에도 비용함수의 최소값을 찾기 위해 경사하강법(Ch7에서 다룸)을 사용하는 경우 경사의 값에 미치는 영향에도 차이가 있습니다. 이는 Ch7에서 다루도록 합니다. [↑](#footnote-ref-8)
8. 관련 코드는 vector\_example.ipynb 파일을 참고하세요. [↑](#footnote-ref-9)
9. Anaconda를 이용해서 파이썬을 설치한 경우에는 numpy가 기본적으로 설치되어 있기 때문에, 추가적으로 설치할 필요가 없습니다. [↑](#footnote-ref-10)
10. numpy에서는 벡터를 1D array로 표현합니다. 여기서 1D는 1차원을 의미하는 것이 아니라 원소들을 일렬로 배열한 array라는 의미를 갖습니다. 참고로 행렬은 원소들을 가로와 세로 양방향으로 배열하기 때문에 2D array로 표현됩니다. [↑](#footnote-ref-11)
11. 선형대수를 의미하는 linear algebra의 줄임말입니다. [↑](#footnote-ref-12)
12. norm은 선형대수에서 벡터의 길이를 의미합니다. [↑](#footnote-ref-13)
13. 승은 제곱근을 의미합니다. [↑](#footnote-ref-14)
14. distance 모듈은 벡터 간의 거리를 계산하는데 사용되는 여러가지 함수를 제공하는데, 이는 <https://docs.scipy.org/doc/scipy/reference/spatial.distance.html>에서 확인할 수 있습니다. [↑](#footnote-ref-15)
15. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.jaccard.html#scipy.spatial.distance.jaccard>를 참고하세요. [↑](#footnote-ref-16)
16. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.hamming.html#scipy.spatial.distance.hamming>를 참고하세요. [↑](#footnote-ref-17)
17. 불확실한 어떤 것을 관찰하는 과정이라고 정의되기도 합니다 (A process by which we observe something uncertain, Pishro-Nik, 2016, p.20). [↑](#footnote-ref-18)
18. 무작위 시행에서는 이러한 표본 공간이 전체 집합이 되는 것입니다. [↑](#footnote-ref-19)
19. 사건 이 발생했다라는 것은 100개의 제품 중에서 결함이 없는 제품 한 개가 선택되었다는 것을 의미합니다. 따라서 남은 제품의 수는 99가 되고, 그 중에 결함이 없는 제품의 수는 94가 됩니다. [↑](#footnote-ref-20)
20. Bc는 B의 여집합을 의미합니다. [↑](#footnote-ref-21)
21. 1은 일반적으로 성공 (success)을 의미하며, 0은 실패 (failure)를 의미합니다. 취하는 값이 반드시 0과 1이 아니라도, 취하는 값이 2개인 변수는 베르누이 변수가 됩니다. 왜냐하면, 취하는 값이 구체적으로 무엇이든 상관없이 그 수가 2개인 경우에는 하나의 값을 0으로 다른 값을 1로 표현할 수 있기 때문입니다. 예를 들어, 감성분석의 경우, 감성을 나타내는 변수가 취하는 값은 긍정과 부정, 두 개이지만 긍정을 1로, 부정을 0으로 표현할 수 있기 때문에 해당 변수도 베르누이 변수로 간주됩니다. [↑](#footnote-ref-22)
22. 변수가 무수히 많은 값들을 취할 수 있기 때문에, 그 중에 하나의 값을 갖을 확률은 0에 가깝다고 생각할 수 있습니다. [↑](#footnote-ref-23)
23. 물리에서의 밀도의 의미와 비슷하다고 생각할 수 있습니다. 밀도는 부피를 구하는데 사용됩니다. 확률에서도 밀도함수를 이용해서 면적을 구하게 됩니다. [↑](#footnote-ref-24)
24. 표준편차를 제곱하면 분산이 됩니다. 따라서 해당 분포의 분산은 σ2입니다. [↑](#footnote-ref-25)
25. 평균은 기대값이라고도 표현합니다. [↑](#footnote-ref-26)
26. 해당 확률밀도함수는 균등 분포 (Uniform distribution)의 확률밀도함수입니다. 이는 로 표현됩니다. 이는 변수 X가 와 사이의 값을 취할 확률이 값이 무엇인지와 상관없이 모두 동일한 분포를 의미합니다. [↑](#footnote-ref-27)
27. 에서 도 X에 대한 함수이므로 가 됩니다. [↑](#footnote-ref-28)
28. 여기서는 연속변수 중심으로 모멘트를 설명합니다. 이산변수의 모멘트도 비슷한 식으로 정의됩니다. [↑](#footnote-ref-29)
29. 이러한 단어들은 일반적으로 불용어가 제거된 특정한 품사의 단어들이 됩니다. [↑](#footnote-ref-30)
30. 나중에 다루게 될 딥러닝의 경우는 단어들의 순서를 고려합니다. 딥러닝에서는 BoW 방법이 사용되지 않습니다. 대신 각 단어들을 특정 차원의 벡터로 변환한 정보를 사용합니다. [↑](#footnote-ref-31)
31. IDF 값을 계산하는데 사용되는 일반적인 공식은 log [ n / (DF + 1) ]입니다. 여기서 n은 텍스트 데이터에 존재하는 전체 문서의 수입니다. [↑](#footnote-ref-32)
32. scikit-learn/싸이킷/을 의미하며 아나콘다를 이용하여 파이썬을 설치하는 경우 기본적으로 설치되어 있습니다. [↑](#footnote-ref-33)
33. 텍스트 데이터에 있는 단어들을 알파벳 순서로 배열했을 때 첫 번째 단어를 의미합니다. [↑](#footnote-ref-34)
34. 컴퓨터에 따라 0.99999999 또는 1.00000001의 결과가 나올 수 있습니다. [↑](#footnote-ref-35)
35. 예제 코드는 docs\_clustering\_examples.ipynb 파일을 참고하세요. [↑](#footnote-ref-36)
36. 군집 ID에 대한 여러분의 결과는 위와 다를 수 있습니다. [↑](#footnote-ref-37)
37. 관련 코드는 docs\_clustering\_examples.ipynb 파일을 참고하세요. [↑](#footnote-ref-38)
38. 해당 생성자 함수 관련 자세한 내용은 <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html> 를 참고하세요. [↑](#footnote-ref-39)
39. 관련 논문은 Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996, August). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd* (Vol. 96, No. 34, pp. 226-231)을 참고하세요. [↑](#footnote-ref-40)
40. 관련 코드는 'docs\_clustering\_examples.ipynb' 를 참고하세요. [↑](#footnote-ref-41)
41. 문서의 벡터화 섹션 참고 [↑](#footnote-ref-42)
42. 확률에 대한 내용은 7장을 참고하세요. [↑](#footnote-ref-43)
43. 정보 이론에 관심이 있는 독자들은 Cover, T. M., & Thomas, J. A. (2012). Elements of information theory. John Wiley & Sons. 을 참고하기 바랍니다. [↑](#footnote-ref-44)
44. 즉 X가 항상 같은 값만을 갖는다는 것을 의미합니다. 이 때 불확실성 = 0 이라고 말할 수 있는 것입니다. [↑](#footnote-ref-45)
45. Kibriya, A. M., Frank, E., Pfahringer, B., & Holmes, G. (2004, December). Multinomial naive bayes for text categorization revisited. In *Australasian Joint Conference on Artificial Intelligence* (pp. 488-499). Springer, Berlin, Heidelberg. [↑](#footnote-ref-46)
46. 가중 평균을 사용할 수도 있습니다. [↑](#footnote-ref-48)
47. 엔트로피라는 개념은 정보 이론 (information theory)에서 사용되는 개념입니다. 엔트로피는 변수의 불확실성 (uncertainty)을 의미합니다. 여기서는 특정 그룹에 존재하는 종속변수의 불확실성이라고 생각할 수 있습니다. [↑](#footnote-ref-49)
48. 데이터는 iris\_data.csv 파일에 저장되어 있습니다 [↑](#footnote-ref-50)
49. DT\_clf\_example.ipynb [↑](#footnote-ref-51)
50. 는 학습률(learning rate)을 고려하는 경우, 와 같이 표현됩니다. [↑](#footnote-ref-52)
51. 원 논문에서는 목적함수 (objective function)이라는 표현을 사용했습니다. [↑](#footnote-ref-53)
52. rbf kernel은 저차원 공간의 벡터를 무한 차원 (infinite dimension)으로 보내고 그곳에서의 내적을 구하는 효과가 있습니다. 자세한 내용은 <https://www.youtube.com/watch?v=XUj5JbQihlU&t=1553s> 을 참고하세요. [↑](#footnote-ref-54)
53. SVM\_sentiment.ipynb 파일을 참고하세요. [↑](#footnote-ref-55)
54. 보다 자세한 내용은 <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html> 을 참고하세요. [↑](#footnote-ref-56)
55. 파이썬 코드는 lexicon\_based\_senti.ipynb를 참고하세요. [↑](#footnote-ref-57)
56. 공개된 영어 감성어 사전의 예: Bing Liu's Opinion Lexicon, SentiWordNet, VADER. 파이썬의 nltk 모듈에서는 SentiWordNet와 VADER 사전을 기반으로한 감성 분석 기능을 제공합니다. [↑](#footnote-ref-58)
57. Hutto, C. J., & Gilbert, E. (2014, May). Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth international AAAI conference on weblogs and social media*. [↑](#footnote-ref-59)
58. 관련 논문은 Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. Journal of the American society for information science, 41(6), 391-407을 참고하세요. [↑](#footnote-ref-60)
59. LSI\_example.ipynb 참고 [↑](#footnote-ref-61)
60. svd\_example.ipynb 파일을 참고하세요. [↑](#footnote-ref-62)
61. PCA와 SVD 모두 고유분해를 기반으로 한 방법들입니다. PCA는 원 데이터가 아니라 공분산 행렬을 이용한다는 것이고, SVD는 원 데이터를 이용한다는 것이 가장 큰 차이입니다. [↑](#footnote-ref-63)
62. 자세한 코드는 LSI\_example.ipynb를 참고하세요. [↑](#footnote-ref-64)
63. gensim 관련 자세한 내용은 <https://radimrehurek.com/gensim/models/lsimodel.html> 을 참고하세요. 파이썬 코드는 LSI\_example.ipynb을 참고하세요. [↑](#footnote-ref-65)
64. gensim은 명령프롬프트 창에서 pip install –upgrade gensim을 통해 설치합니다. [↑](#footnote-ref-66)
65. LSI\_example.ipynb 파일의 build\_doc\_word\_matrix() 함수 참고 [↑](#footnote-ref-67)
66. 해당 코드는 LSI\_gensim\_real\_data.ipynb를 참고하세요. [↑](#footnote-ref-68)
67. Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. Journal of machine Learning research, 3(Jan), 993-1022. [↑](#footnote-ref-69)
68. Blei, D. M. (2012). Probabilistic topic models. Communications of the ACM, 55(4), 77-84. [↑](#footnote-ref-70)
69. 깁스 샘플링에 대한 설명인 본 책의 범위를 벗어나므로 여기에서는 설명하지 않겠습니다. 관심있는 독자는 Kruschke, J. (2014). Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan. Academic Press. 또는 Donovan, T. M., & Mickey, R. M. (2019). Bayesian Statistics for Beginners: A Step-by-step Approach. Oxford University Press. 와 같은 책을 참고 바랍니다. [↑](#footnote-ref-71)
70. 보다 쉬운 설명은 Zhai, C., & Massung, S. (2016). Text data management and analysis: a practical introduction to information retrieval and text mining. Association for Computing Machinery and Morgan & Claypool.을 참고하세요. 최대우도추정 방법이 아니라, EM (Expectation-Maximization)추정 방법이 사용되기도 합니다. EM 방법에 대한 설명도 본 책의 범위를 벗어나므로 관심있는 독자는 Pawitan, Yudi. In all likelihood: statistical modelling and inference using likelihood. Oxford University Press, 2001. 또는 http://cs229.stanford.edu/notes/cs229-notes8.pdf 을 참고하세요. [↑](#footnote-ref-72)
71. 해당 코드는 LDA\_example.ipynb을 참고하세요. 해당 코드는 2016년에 주요 신문사에서 발간된 정치, 사회, 경제 관련 기사 7,837에 대한 것입니다. [↑](#footnote-ref-73)
72. 관련 논문은 <http://svn.aksw.org/papers/2015/WSDM_Topic_Evaluation/public.pdf> 또는 Röder, M., Both, A., & Hinneburg, A. (2015, February). Exploring the space of topic coherence measures. In Proceedings of the eighth ACM international conference on Web search and data mining (pp. 399-408)을 참고하세요. [↑](#footnote-ref-74)